ROBERT E. KREBS, CA BAR NO. 57526
rkrebs@thelen.com
CHRISTOPHER L. OGDEN, CA BAR NO. 235517
cogden@thelen.com
THELEN REID BROWN RAYSMAN & STEINER LLP
225 West Santa Clara Street, Suite 1200
San Jose, CA 95113-1723
Tel. 408.292.5800
Fax 408.287.8040

RONALD F. LOPEZ, CA BAR NO. 111756
rflopez@thelen.com
SUSHILA CHANANA, CAR BAR NO. 254100
schanana@thelen.com
THELEN REID BROWN RAYSMAN & STEINER LLP
101 Second Street, Suite 1800
San Francisco, CA 94105-3606
Tel. 415.371.1200
Fax. 415.371.1211

Attorneys for Defendants TECHNOLOGY PROPERTIES LIMITED and ALLIACENSE LIMITED.

CHARLES T. HOGE, CA BAR NO. 110696
choge@knlh.com
KIRBY NOONAN LANCE & HOGE, LLP
350 Tenth Avenue, Suite 1300
San Diego, CA 92101
Tel. 619.231.8666, Fax. 619.231.9593

Attorneys for Defendant PATRIOT SCIENTIFIC CORPORATION

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN JOSE DIVISION

HTC CORPORATION and HTC AMERICA, INC.,

Plaintiffs,

v.

TECHNOLOGY PROPERTIES LIMITED, PATRIOT SCIENTIFIC CORPORATION, and ALLIACENSE LIMITED,

Defendants.

CASE NO. 08-CV-00882 JF

DECLARATION OF SUSHILA CHANANA IN SUPPORT OF DEFENDANTS' MOTION

Date: August 1, 2008
Time: 9:00 a.m.
Place: Courtroom 3, 5th Floor
Judge: Hon. Jeremy Fogel

I, Sushila Chanana, declare and state as follows:

1. I am an attorney with the law firm of Thelen Reid Brown Raysman & Steiner LLP, counsel of record to Technology Properties Limited. I have personal knowledge of the matters set forth in this declaration, and if called as a witness, I could and would testify competently thereto. This declaration is submitted in support of the previously-filed Motion to Dismiss, or in the Alternative, to Transfer Venue, or in the Alternative, To Stay ("Motion").

2. Upon information and belief, Charles H. Moore, an inventor listed on the Moore Microprocessor Portfolio ("MMP") patents,[1] resides in Nevada.

3. Attached as **Exhibit A** is a true and correct copy of information obtained from the Texas Secretary of State website, http://www.sos.state.tx.us/corp/index.shtml, accessed on July 18, 2008.

4. Attached as **Exhibit B** is a true and correct copy of this document obtained from the Texas Secretary of State website, http://www.sos.state.tx.us/corp/index.shtml from the website, accessed on July 18, 2008. The document, entitled "Texas Franchise Tax Public Information Report," filed on December 31, 2006, states that the "principal office" for HTC USA, Inc. (a prior name for HTC America, Inc., *see* Ex. A) is 5950 Corporate Dr., Houston, TX 77036-2306.

5. Attached as **Exhibit C** is a true and correct copy of a document from the Texas Secretary of State website, http://www.sos.state.tx.us/corp/index.shtml, accessed on July 18, 2008, entitled, "Second Amendment to the Articles of Incorporation," filed on July 9, 2007, which states that HTC America, Inc., is a "corporation incorporated and existing under and by virtue of the

[1] The MMP patents include U.S. patent Nos., 5,440,749, 5,809,336, 5,784,854, and 5,530,890.

provisions of the Texas business Corporation Act" and that it was originally incorporated in Texas on January 6, 2003.

6. Attached as **Exhibit D** is a true and correct copy of a document obtained from the Texas Secretary of State website, http://www.sos.state.tx.us/corp/index.shtml, accessed on July 18, 2008, which lists HTC America, Inc.'s current "Entity Status" in Texas is "In existence."

7. Attached as **Exhibit E** is a true and correct copy of *Technology Properties Limited, and Patriot Scientific Corporation v. HTC Corporation and HTC America, Inc.*, Case No., 2:08-CV-00172-DF, filed in the Eastern District of Texas ("E. D. of Texas").

8. Attached as **Exhibit F** is a true and correct copy of *Technology Properties Limited, and Patriot Scientific Corporation v. HTC Corporation, HTC America, Inc. and Asustek Computer, Inc.*, Case No., 2:08-CV-00174-TJW, filed in the E. D. of Texas on April 25, 2008.

9. Attached as **Exhibit G** is a true and correct copy of *Technology Properties Limited and Patriot Scientific Corporation vs. HTC Corporation and HTC America, Inc.*, Case No., 2:08-cv-226-TJW, filed in the E. D. of Texas on June 4, 2008.

I declare under penalty of perjury under the laws of the United States of America the foregoing statements are true and correct; and all statements of belief are believed to be true and correct.

EXECUTED this 18 day of July 2008 in San Francisco, California.

/s/ Sushila Chanana
Sushila Chanana

Exhibit A

Texas Secretary of State

UCC | Business Organizations | Trademarks | Account | Help/Fees | Briefcase | Logout

BUSINESS ORGANIZATIONS INQUIRY - VIEW ENTITY

Filing Number:	800160088	**Entity Type:**	Domestic For-Profit Corporation
Original Date of Filing:	January 6, 2003	**Entity Status:**	In existence
Formation Date:	N/A		
Tax ID:	18205886890	**FEIN:**	
Duration:	Perpetual		

Name: HTC America, Inc.
Address: 5950 CORPORATE DR
Houston, TX 77036-2306 USA

REGISTERED AGENT | FILING HISTORY | NAMES | MANAGEMENT | ASSUMED NAMES | ASSOCIATED ENTITIES

Name	Name Status	Name Type	Name Inactive Date	Consent Filing #
HTC USA, Inc.	Prior	Legal	August 1, 2006	0
HTC America, Inc.	In use	Legal		

Order | Return to Search

Instructions:
- To place an order for additional information about a filing press the 'Order' button.

Exhibit B

Comptroller of Public Accounts FORM 05 - 102 (Rev.1-05/24)

3333

b.

13196 a. TCode

Do not write in the space above

TEXAS FRANCHISE TAX PUBLIC INFORMATION REPORT

MUST be filed to satisfy franchise tax requirements

Corporation name and address

HTC USA, INC.
5950 CORPORATE DR.
HOUSTON TX 77036-2306

c. Taxpayer identification number	d. Report year
18205886890	2006

e. PIR / IND ☐ 1 ☐ 4

Secretary of State file number or, if none, Comptroller unchartered number
g. *Item k on Franchise Tax Report Form 05-142* 0800160088

If the preprinted information is not correct, please type or print the correct information.

The following information MUST be provided for the Secretary of State (SOS) by each corporation or limited liability company that files a Texas Corporation Franchise Tax Report. Use additional sheets for Sections A, B, and C, if necessary. The information will be available for public inspection.

☐ Check here if there are currently no changes to the information preprinted in Section A of this report. Then, complete Sections B and C.

*18205886890006

Please sign below! Officer and director information is reported as of the date a Public Information Report is completed. The information is updated annually as part of the franchise tax report. There is no requirement or procedure for supplementing the information as officers and directors change throughout the year.

Corporation's principal office
5950 CORPORATE DR., HOUSTON, TX 77036-2306

Principal place of business
5950 CORPORATE DR., HOUSTON, TX 77036-2306

SECTION A. Name, title, and mailing address of each officer and director.

NAME	TITLE	DIRECTOR	Term expiration (mm-dd-yyyy)
FRED LIU	OFFICER	☐ YES	
MAILING ADDRESS: SAME AS ABOVE			
NAME	TITLE	DIRECTOR	Term expiration (mm-dd-yyyy)
		☐ YES	
MAILING ADDRESS			
NAME	TITLE	DIRECTOR	Term expiration (mm-dd-yyyy)
		☐ YES	
MAILING ADDRESS			
NAME	TITLE	DIRECTOR	Term expiration (mm-dd-yyyy)
		☐ YES	
MAILING ADDRESS			
NAME	TITLE	DIRECTOR	Term expiration (mm-dd-yyyy)
		☐ YES	
MAILING ADDRESS			

SECTION B. List each corporation or limited liability company, if any, in which this reporting corporation or limited liability company owns an interest of ten percent (10%) or more. Enter the information requested for each corporation or limited liability company.

Name of owned (subsidiary) corporation	State of incorporation	Texas SOS file number	Percentage interest
Name of owned (subsidiary) corporation	State of incorporation	Texas SOS file number	Percentage interest

SECTION C. List each corporation or limited liability company, if any, that owns an interest of ten percent (10%) or more in this reporting corporation or limited liability company. Enter the information requested for each corporation or limited liability company.

Name of owning (parent) corporation	State of incorporation	Texas SOS file number	Percentage interest

Registered agent and registered office currently on file. *(See instructions if you need to make changes.)*

Agent:

Office:

☐ Check here if you need forms to change this information. Changes can also be made on-line at *http://www.sos.state.tx.us/corp/sosda/index.shtml*

I declare that the information in this document and any attachments is true and correct to the best of my knowledge and belief, as of the date below, and that a copy of this report has been mailed to each person named in this report who is an officer or director and who is not currently employed by this corporation or limited liability company or a related corporation.

sign here ▶ Officer, director, or other authorized person	Title	Date	Daytime phone (Area code and number)
	Secretary	9/12	(832) 228-8839

5Y5212 2.000 THO

Exhibit C

SECOND AMENDMENT TO THE ARTICLES OF INCORPORATION

OF

HTC America, Inc.

FILED
In the Office of the
Secretary of State of Texas
JUL 09 2007
Corporations Section

(Pursuant to Articles 4.01-4.06 of the Texas Business Corporation Act)

HTC America, Inc., a corporation incorporated and existing under and by virtue of the provisions of the Texas Business Corporation Act,

DOES HEREBY CERTIFY:

FIRST: That the name of this corporation is HTC America, Inc. and that this corporation was originally incorporated pursuant to the Texas Business Corporation Act on January 6, 2003.

SECOND: That an Articles of Amendment was executed on August 2, 2006.

THIRD: That the Board of Directors duly adopted resolutions proposing to amend the Articles of Incorporation of this corporation as amended, declaring said second amendment to be advisable and in the best interests of this corporation and its stockholders, and authorizing the appropriate officers of this corporation to solicit the consent of the stockholders therefore, which resolution setting forth the proposed amendment is as follows:

RESOLVED, that the Articles of Incorporation of this corporation be amended as follows:

ARTICLE 4

The total number of shares that the corporation is authorized to issue is thirty million (30,000,000) shares and the par value of each of the authorized shares is $1.00.

The remaining content of this Article 4 in the Articles of Incorporation shall stay unchanged.

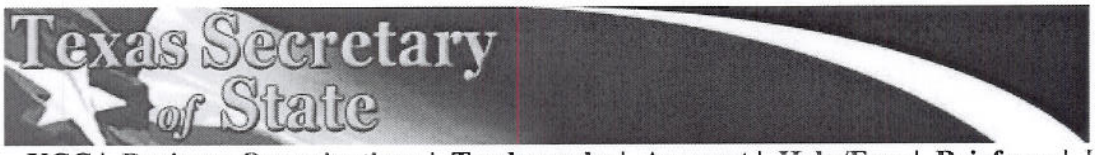
FOURTH: The foregoing amendment was approved by the unanimous written consent of the sole shareholder of the corporation on December 28, 2006.

FIFTH: That said amendment was duly adopted in accordance with the provisions of the Texas Business Corporation Act.

IN WITNESS WHEREOF, this Amendment to the Articles of Incorporation, as amended, has been executed by the authorized representative of this corporation on this 15th day of June, 2007.

Fred Liu, President

Exhibit D



FIND ENTITY NAME SEARCH

This search was performed on with the following search parameter:
ENTITY NAME : htc america

Mark	Filing Number	Name	Entity Type	Entity Status	Name Type	Name Status
○	800160088	HTC America, Inc.	Domestic For-Profit Corporation	In existence	Legal	In use
○	4411010	HTC, LTD.	Domestic Limited Partnership (LP)	Cancelled	Legal	Inactive
○	16892900	HTC COMPANY	Domestic For-Profit Corporation	Voluntarily dissolved	Legal	Inactive
○	22777600	H. T. C. INC.	Domestic For-Profit Corporation	Forfeited existence	Legal	Inactive
○	33615500	H.T.C., INC.	Domestic For-Profit Corporation	Merged	Legal	Inactive
○	55998800	HTC, INC.	Domestic For-Profit Corporation	Forfeited existence	Legal	Inactive
○	66997100	H.T.C., INC.	Domestic For-Profit Corporation	Voluntarily dissolved	Legal	Inactive
○	146872600	HTC CORPORATION	Domestic For-Profit Corporation	Merged	Legal	Inactive
○	800285428	HTCS, Inc.	Domestic For-Profit Corporation	Forfeited existence	Legal	Inactive
○	800832278	HTC, Inc.	Foreign For-Profit Corporation	In existence	Legal	In use

Records 1 to 10 of 39 scroll Next >> **OR** proceed to page [ ] of 4 pages GO

Return to Order | New Search

Instructions:
- To view additional information pertaining to a particular filing select the number associated with the name.
- To place an order for additional information about a filing select the radial button listed under 'Mark' that is associated with the entity and press the 'Order' button.

EXHIBIT E

IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

(1) TECHNOLOGY PROPERTIES LIMITED and (2) PATRIOT SCIENTIFIC CORPORATION,

Plaintiffs,

vs

(1) HTC CORPORATION and (2) HTC AMERICA, INC.,

Defendants.

CASE NO. 2-08CV-172 DF

Jury Trial Demanded

COMPLAINT FOR PATENT INFRINGEMENT AND DEMAND FOR JURY TRIAL

Plaintiffs, Technology Properties Limited ("TPL") and Patriot Scientific Corporation ("Patriot"), (collectively "Plaintiffs"), allege the following in support of their Complaint for Patent Infringement and Demand for Jury Trial ("Complaint") against Defendants, HTC Corporation ("HTC") and HTC America, Inc. ("HTC America")

PARTIES

1. Plaintiff, Technology Properties Limited ("TPL") is a corporation duly organized and existing under the laws of the State of California and maintains its principal place of business in Cupertino, California.

2. Plaintiff, Patriot Scientific Corporation ("Patriot") is a corporation duly organized and existing under the laws of the State of Delaware and maintains its principal place of business in Carlsbad, California

3 Upon information and belief, Defendant HTC Corporation is a Taiwan corporation with its principal place of business in Taoyuan, Taiwan, R.O.C.

4 Upon information and belief, Defendant HTC America, Inc. is a Texas corporation with its principal place of business in Bellevue, Washington.

JURISDICTION

5 This Court has subject matter jurisdiction over this action pursuant to 28 U.S.C. §§ 1331, 1338(a) because this action arises under the patent laws of the United States, including 35 U.S.C. §§ 101, *et seq.* and 271, *et seq.* This Court has personal jurisdiction over Defendants because they each infringe Plaintiffs' patents by offering on their websites infringing products to their users and/or customers who reside in, or may be found in, the Eastern District of Texas. Further, each Defendant has actually transacted business with users of their websites in the Eastern District of Texas.

VENUE

6. Venue is proper in this judicial district under 28 U.S.C. §§ 1391(b) and 1400(b) because Defendants reside in this district, have each committed acts of infringement in this district and, through their websites, have a regular and established place of business in this district.

GENERAL ALLEGATIONS

7 On September 15, 1998, United States Patent No. 5,809,336 ('336 Patent") entitled "High Performance Microprocessor Having Variable Speed System Clock" was duly and legally issued. All rights and interest in the '336 Patent are co-owned by TPL and Patriot Scientific Corporation. TPL has the sole and exclusive right and obligation to license and

enforce the '336 Patent. A true and correct copy of the '336 Patent is attached hereto as Exhibit A.

8. On August 8, 1995, United States Patent No. 5,440,749 ("749 Patent") entitled "High Performance, Low Cost Microprocessor Architecture" was duly and legally issued. All rights and interest in the '749 Patent are co-owned by TPL and Patriot. TPL has the sole and exclusive right and obligation to license and enforce the '749 Patent. A true and correct copy of the '749 Patent is attached hereto as Exhibit B.

9. On July 22, 2003, United States Patent No. 6,598,148 ('148 Patent") entitled "High Performance Microprocessor Having Variable Speed System Clock" was duly and legally issued. All rights and interest in the '148 Patent are co-owned by TPL and Patriot. TPL has the sole and exclusive right and obligation to license and enforce the '148 Patent. A true and correct copy of the '148 Patent is attached hereto as Exhibit C.

COUNT 1

(Patent Infringement Against HTC Corporation)

10. Paragraphs 1-9 of the Complaint set forth above are incorporated herein by reference.

11. Upon information and belief, Defendant HTC has infringed and continues to infringe under 35 U.S.C. § 271 the '336 Patent, '749 Patent, and '148 Patent (collectively, "patents-in-suit").

12. HTC's acts of infringement have caused damage to Plaintiffs. Under 35 U.S.C. § 284, TPL and Patriot are entitled to recover from HTC the damages sustained by Plaintiffs as a result of its infringement of the patents-in-suit. HTC's infringement of Plaintiffs' exclusive rights under the patents-in-suit will continue to damage Plaintiffs' business, causing irreparable

harm, for which there is no adequate remedy at law, unless enjoined by this Court under 35 U.S.C. § 283.

13. Plaintiffs allege, on information and belief, that HTC's acts of infringement were willful and deliberate.

COUNT 2

(Patent Infringement Against HTC America, Inc.)

14. Paragraphs 1-9 of the Complaint set forth above are incorporated herein by reference.

15. Upon information and belief Defendant HTC America has infringed and continues to infringe under 35 U.S.C. § 271 the '336 Patent, '749 Patent , and '148 Patent (collectively "patents-in-suit).

16. HTC America's acts of infringement have caused damage to Plaintiffs. Under 35 U.S.C. § 284, TPL and Patriot are entitled to recover from HTC America the damages sustained by Plaintiffs as a result of its infringement of the patents-in-suit. HTC America's infringement of Plaintiffs' exclusive rights under the patents-in-suit will continue to damage Plaintiffs' business, causing irreparable harm, for which there is no adequate remedy at law, unless enjoined by this Court under 35 U.S.C. § 283.

17. Plaintiffs allege, on information and belief, that HTC America's acts of infringement were willful and deliberate.

PRAYER FOR RELIEF

WHEREFORE, Plaintiffs respectfully request that this Court enter judgment against Defendants as follows:

A. For judgment that Defendants HTC Corporation and HTC America, Inc. have

infringed and continue to infringe the patents-in-suit;

B For permanent injunctions under 35 U.S.C. § 283 against Defendants and their directors, officers, employees, agents, subsidiaries, parents, attorneys, and all persons acting in concert, on behalf of, in joint venture, or in partnership with Defendants from further acts of infringement;

C For damages to be paid by Defendants adequate to compensate Plaintiffs for their infringement, including interests, costs and disbursements as the Court may deem appropriate under 35 U.S.C. § 284;

D For judgment finding that Defendants infringement was willful and deliberate, entitling Plaintiffs to increased damages under 35 U.S.C. § 284;

E. For judgment finding this to be an exceptional case against Defendants and awarding Plaintiffs attorney fees under 35 U.S.C. § 285; and,

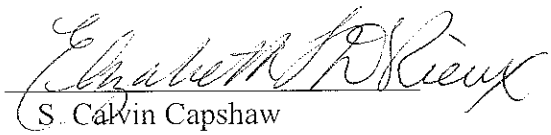
F. For such other and further relief at law and in equity as the court may deem just and proper.

DEMAND FOR JURY TRIAL

Pursuant to the Federal Rules of Civil Procedure Rule 38, Plaintiffs hereby demand a jury trial on all issues triable by jury.

Dated: April 25, 2008

Respectfully submitted,

By: 

S. Calvin Capshaw
State Bar No. 03783900
Email: ccapshaw@capshawlaw.com
Elizabeth L. DeRieux
State Bar No. 05770585
Email: ederieux@capshawlaw.com
Capshaw DeRieux, LLP
1127 Judson Road, Suite 220
Longview, TX 75601
Telephone: (903) 236-9800
Facsimile: (903) 236-8787

Robert E. Krebs
California Bar No. 57526
Email: rkrebs@thelen.com
Christopher L. Ogden
California Bar No. 235517
Email: cogden@thelen.com
Thelen Reid Brown Raysman & Steiner, LLP
225 West Santa Clara Street, Suite 1200
San Jose, CA 95113-1723
Telephone: (408) 292-5800
Facsimile: (408) 287-8040

Ronald F. Lopez
California Bar No. 11756
Email: rflopez@thelen.com
Thelen Reid Brown Raysman & Steiner, LLP
101 Second Street, Suite 1800
San Francisco, CA 94105–3606
Telephone: (415) 371-1200
Facsimile: (415) 371-1211

ATTORNEYS FOR PLAINTIFF
TECHNOLOGY PROPERTIES LIMITED

By: Charles T Hoge by permission EJD

Robert M. Parker
State Bar No. 15498000
Email: rmparker@pbatyler.com
Robert Christopher Bunt
State Bar No. 00787165
Email: rcbunt@pbatyler.com
Parker, Bunt & Ainsworth, P.C.
100 East Ferguson, Ste. 1114
Tyler, TX 75702
Telephone: (903) 531-3535
Facsimile: (903) 533-9687

Charles T. Hoge
California Bar No. 110696
Email: choge@knlh.com
Kirby Noonan Lance & Hoge, LLP
350 Tenth Avenue, Suite 1300
San Diego, CA 92101
Telephone: (619) 231-8666
Facsimile: (619) 231-9593

ATTORNEYS FOR PLAINTIFF
PATRIOT SCIENTIFIC CORPORATION

US005809336A

United States Patent [19]

Moore et al.

[11] Patent Number: 5,809,336

[45] Date of Patent: Sep. 15, 1998

[54] **HIGH PERFORMANCE MICROPROCESSOR HAVING VARIABLE SPEED SYSTEM CLOCK**

[75] Inventors: **Charles H. Moore**, Woodside; **Russell H. Fish, III**, Mt View, both of Calif

[73] Assignee: **Patriot Scientific Corporation**, San Diego, Calif

[21] Appl. No.: **484,918**

[22] Filed: **Jun. 7, 1995**

Related U.S. Application Data

[62] Division of Ser. No. 389,334, Aug. 3, 1989, Pat. No. 5,440,749

[51] **Int. Cl.**[6] **G06F 1/04**

[52] **U.S. Cl.** **395/845**

[58] **Field of Search** 395/500, 551, 395/555, 845

[56] **References Cited**

U.S. PATENT DOCUMENTS

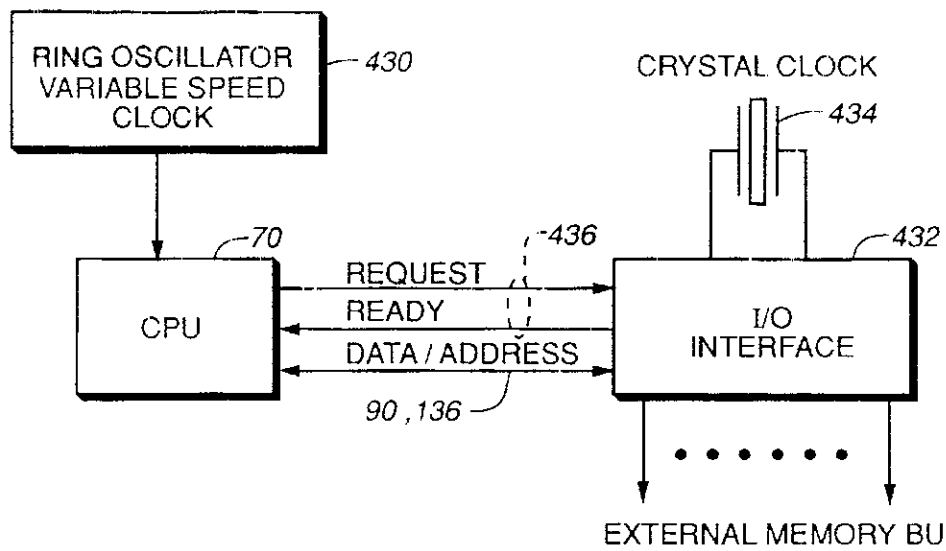
3,967,104	6/1976	Brantingham	364/709.09
3,980,993	9/1976	Bredart et al	395/550
4,003,028	1/1977	Bennett et al	395/742
4,042,972	8/1977	Gruner et al	395/389
4,050,096	9/1977	Bennett	395/494
4,112,490	9/1978	Pohlman et al	395/287
4,315,308	2/1982	Jackson	395/853
4,338,675	7/1982	Palmer	364/748
4,398,265	8/1983	Puhl et al	395/882
4,453,229	6/1984	Schaire	395/250
4,503,500	3/1985	Magan	395/800
4,539,655	9/1985	Trussell et al	395/280
4,553,201	11/1985	Pollack	395/183.22
4,627,082	12/1986	Pelgrom et al	377/63
4,670,837	6/1987	Sheets	395/550
4,680,698	7/1987	Edwards et al	395/800
4,761,763	8/1988	Hicks	395/286
5,414,862	5/1995	Suzuki et al	395/750

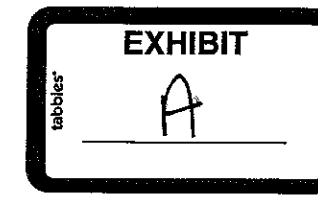
Primary Examiner—David Y. Eng
Attorney, Agent, or Firm—Cooley Godward LLP

[57] **ABSTRACT**

A high performance, low cost microprocessor system having a variable speed system clock is disclosed herein. The microprocessor system includes an integrated circuit having a central processing unit and a ring oscillator variable speed system clock for clocking the microprocessor. The central processing unit and ring oscillator variable speed system clock each include a plurality of electronic devices of like type, which allows the central processing unit to operate at a variable processing frequency dependent upon a variable speed of the ring oscillator variable speed system clock. The microprocessor system may also include an input/output interface connected to exchange coupling control signals, address and data with the central processing unit. The input/output interface is independently clocked by a second clock connected thereto.

10 Claims, 19 Drawing Sheets





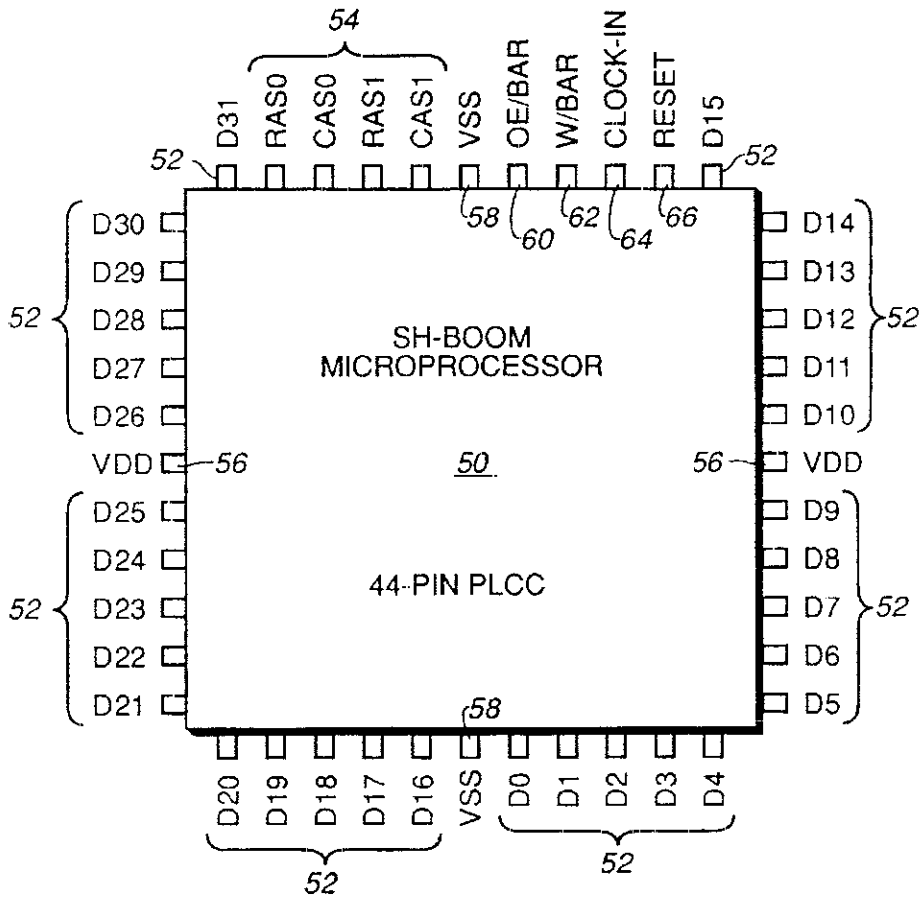

54
D31
RAS0
CAS0
RAS1
CAS1
VSS
OE/BAR
W/BAR
CLOCK-IN
RESET
D15
52
58
60
62
64
66
D30
D29
D28
D27
D26
VDD
56
D14
D13
D12
D11
D10
SH-BOOM
MICROPROCESSOR
50
D25
D24
D23
D22
D21
D9
D8
D7
D6
D5
44-PIN PLCC
D20
D19
D18
D17
D16
D0
D1
D2
D3
D4

FIG._1

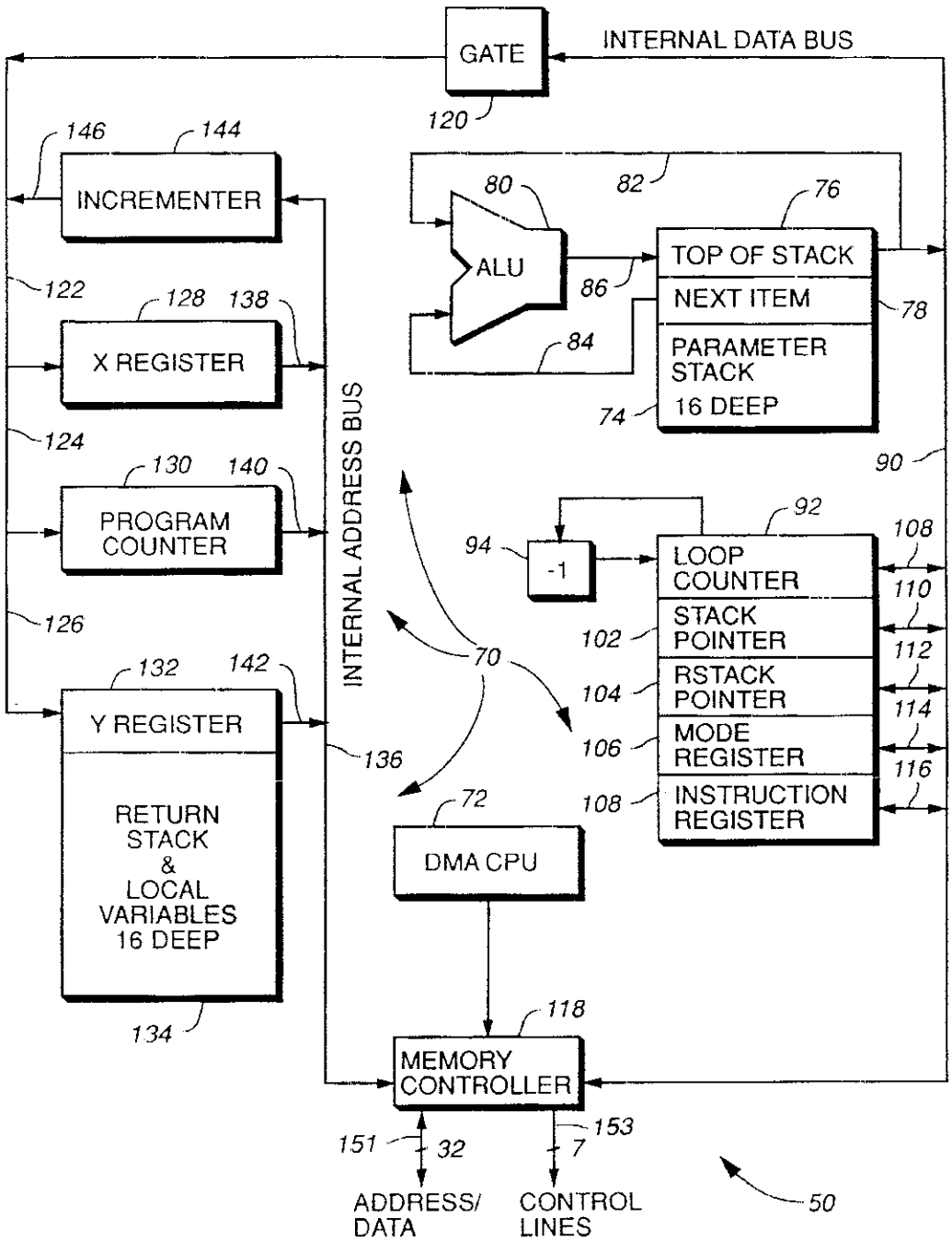

GATE
INTERNAL DATA BUS
120
146
144
INCREMENTER
80
82
76
ALU
TOP OF STACK
NEXT ITEM
PARAMETER STACK
16 DEEP
86
84
78
74
122
128
138
X REGISTER
124
130
140
PROGRAM COUNTER
126
INTERNAL ADDRESS BUS
90
92
94
-1
LOOP COUNTER
STACK POINTER
RSTACK POINTER
MODE REGISTER
INSTRUCTION REGISTER
108
110
112
114
116
102
104
106
108
70
132
142
Y REGISTER
136
RETURN STACK & LOCAL VARIABLES 16 DEEP
134
72
DMA CPU
118
MEMORY CONTROLLER
151
32
153
7
ADDRESS/ DATA
CONTROL LINES
50

FIG._2

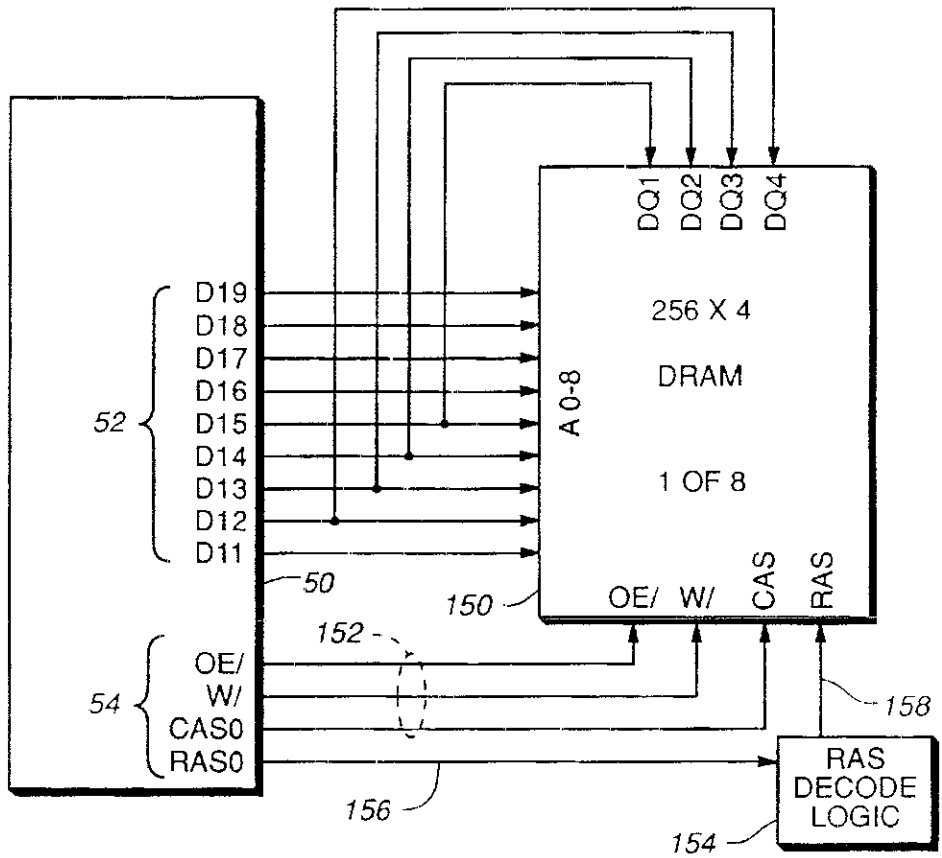

DQ1
DQ2
DQ3
DQ4
256 X 4
DRAM
1 OF 8
A 0-8
OE/
W/
CAS
RAS
D19
D18
D17
D16
D15
D14
D13
D12
D11
52
50
150
152
OE/
W/
CAS0
RAS0
54
156
158
RAS
DECODE
LOGIC
154

FIG._3

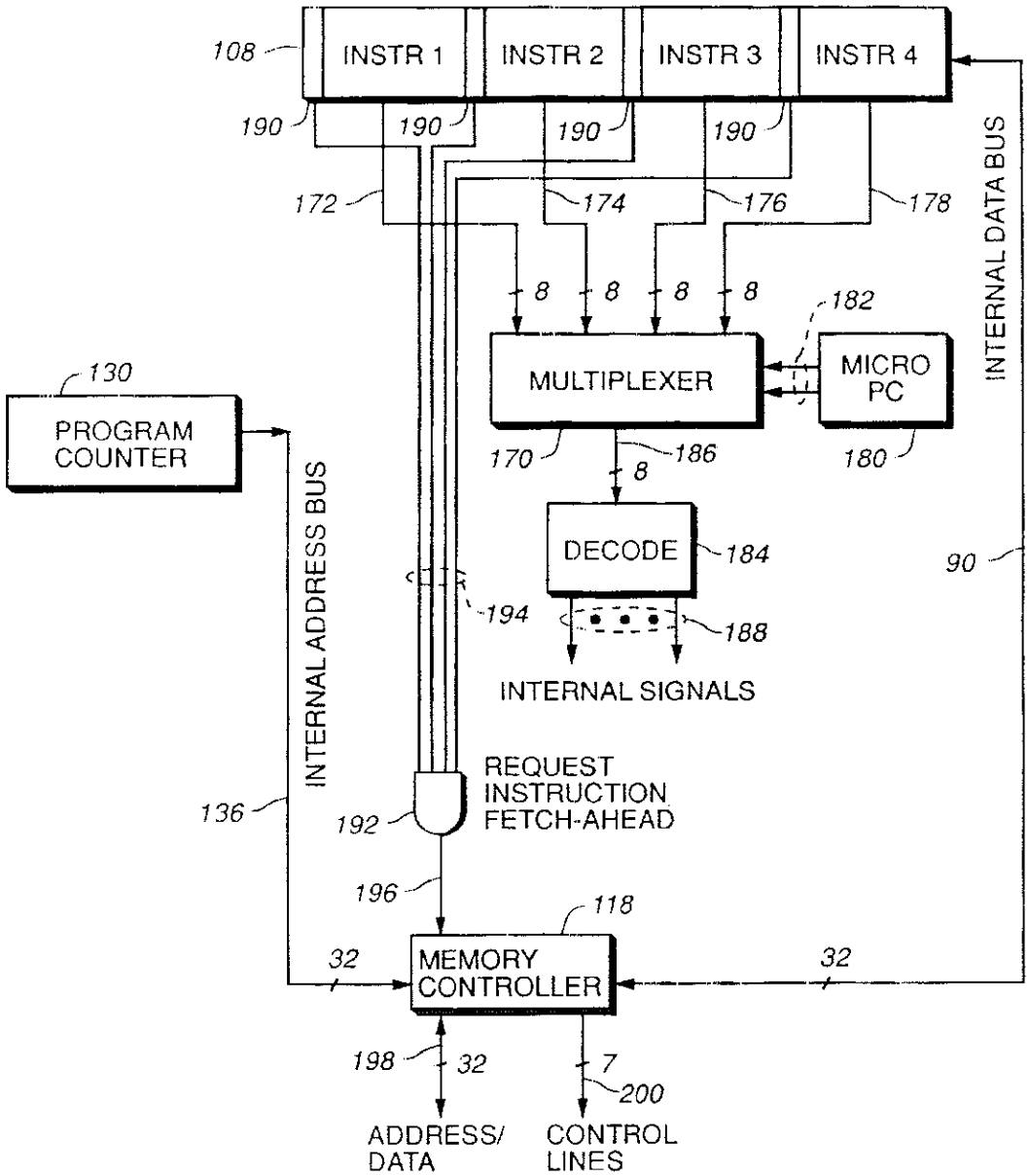

108
INSTR 1
INSTR 2
INSTR 3
INSTR 4
190
190
190
190
172
174
176
178
8
8
8
8
182
MULTIPLEXER
MICRO PC
130
PROGRAM COUNTER
170
186
180
8
DECODE
184
90
194
188
INTERNAL SIGNALS
INTERNAL ADDRESS BUS
INTERNAL DATA BUS
REQUEST INSTRUCTION FETCH-AHEAD
136
192
196
118
32
MEMORY CONTROLLER
32
198
32
7
200
ADDRESS/ DATA
CONTROL LINES

FIG._4

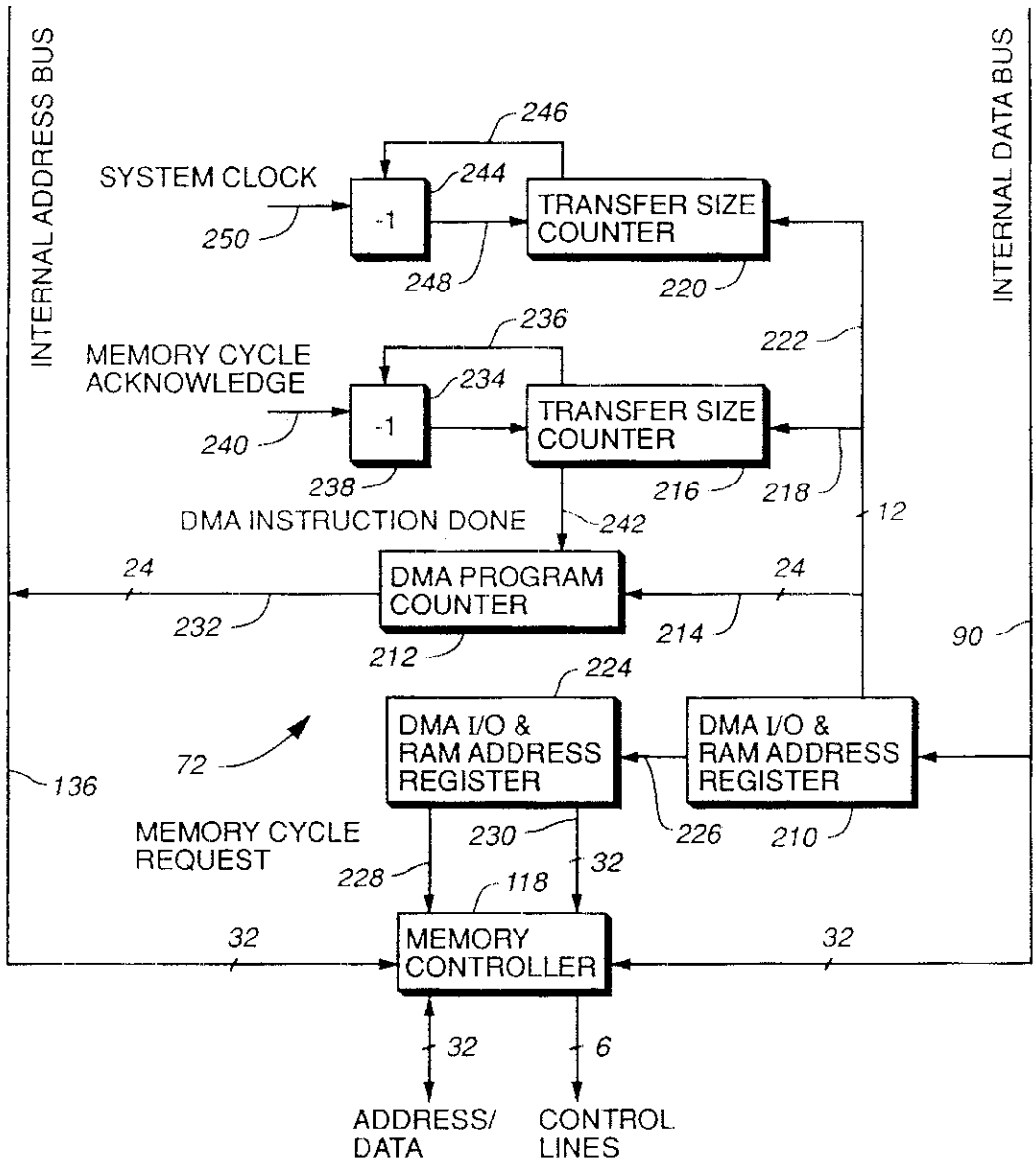

INTERNAL ADDRESS BUS
INTERNAL DATA BUS
SYSTEM CLOCK
MEMORY CYCLE ACKNOWLEDGE
DMA INSTRUCTION DONE
MEMORY CYCLE REQUEST
-1
-1
TRANSFER SIZE COUNTER
TRANSFER SIZE COUNTER
DMA PROGRAM COUNTER
DMA I/O & RAM ADDRESS REGISTER
DMA I/O & RAM ADDRESS REGISTER
MEMORY CONTROLLER
ADDRESS/ DATA
CONTROL LINES
246
244
250
248
220
236
234
240
238
222
216
218
12
242
24
24
232
214
212
90
224
72
136
230
226
210
228
32
118
32
32
32
6

FIG._5

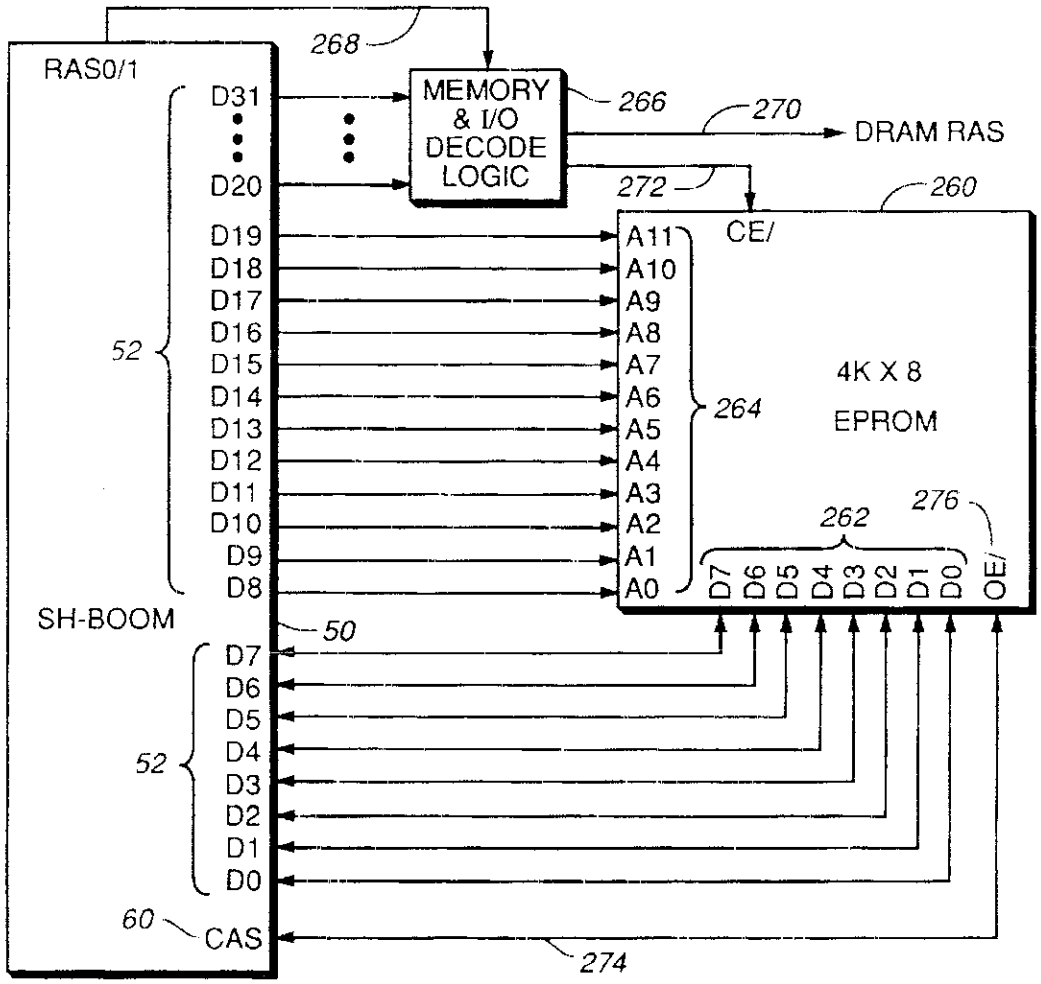

RAS0/1
D31
D20
D19
D18
D17
D16
D15
D14
D13
D12
D11
D10
D9
D8
52
SH-BOOM
50
D7
D6
D5
D4
D3
D2
D1
D0
52
60
CAS
268
MEMORY & I/O DECODE LOGIC
266
270
DRAM RAS
272
260
CE/
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
264
4K X 8
EPROM
262
276
D7
D6
D5
D4
D3
D2
D1
D0
OE/
274

FIG._6

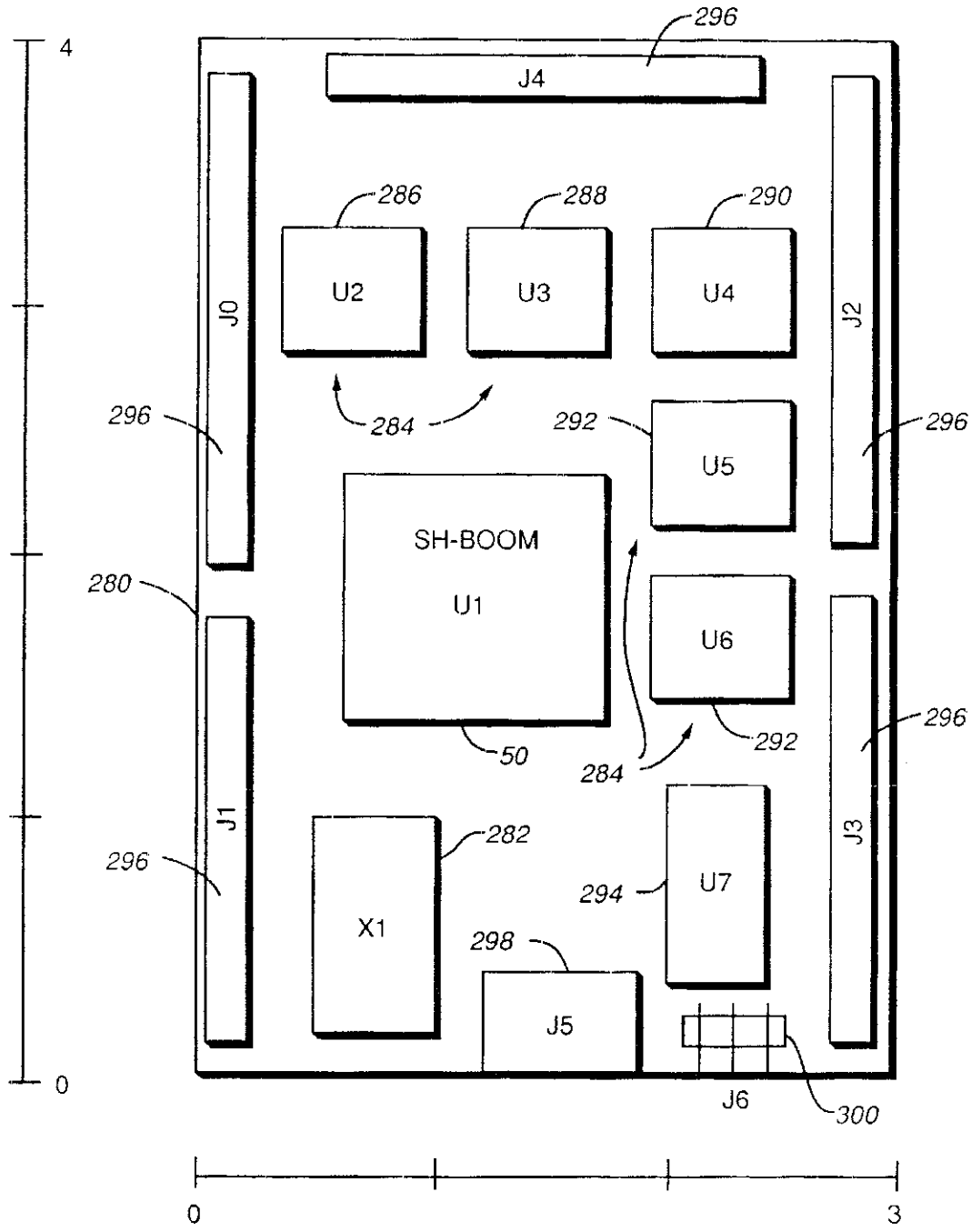

296
J4
4
J0
286
U2
288
U3
290
U4
J2
296
284
292
U5
296
SH-BOOM
U1
280
U6
296
50
284
292
J1
J3
296
282
294
U7
X1
298
J5
J6
300
0
0
3

FIG._7

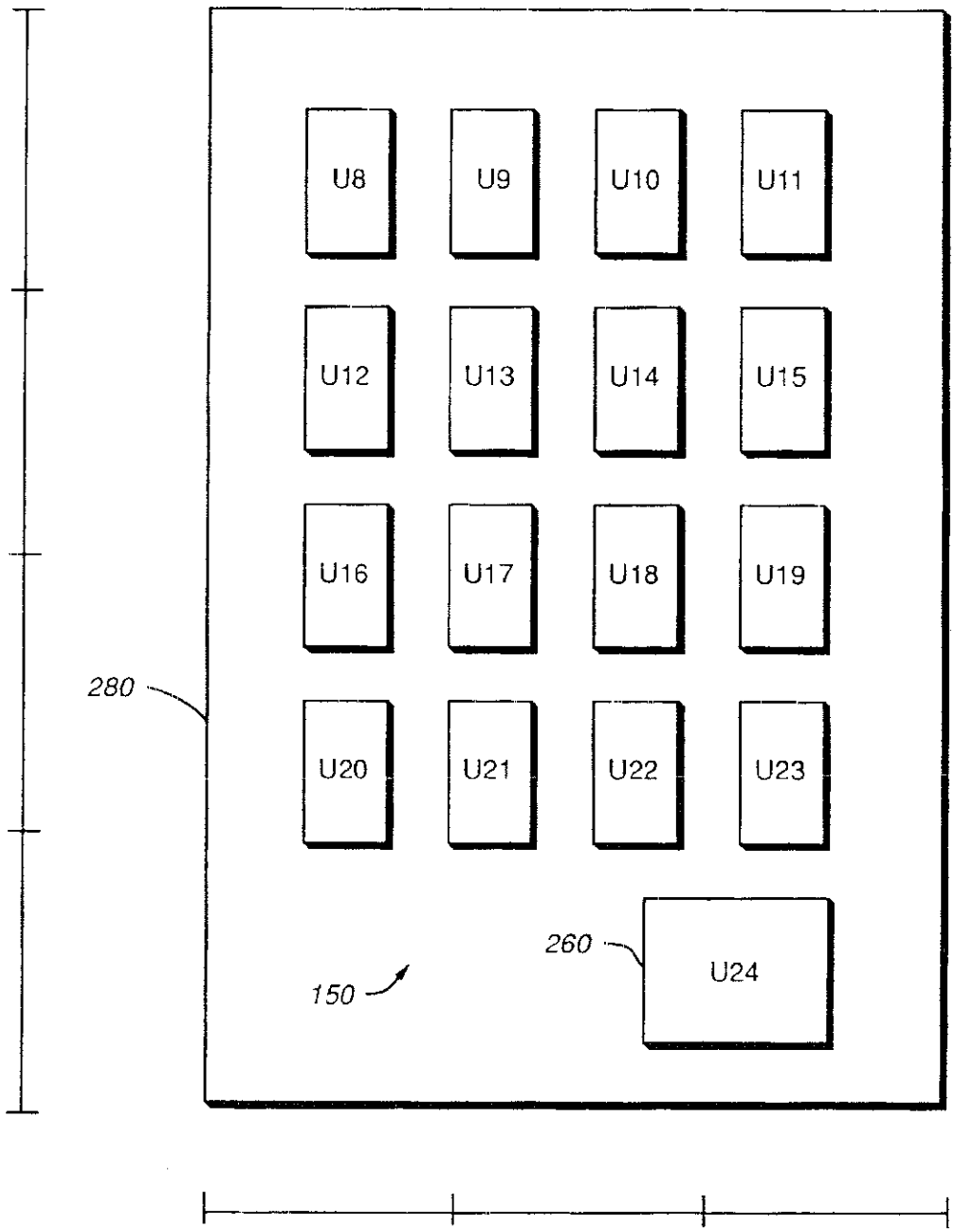

U8
U9
U10
U11
U12
U13
U14
U15
U16
U17
U18
U19
U20
U21
U22
U23
U24
280
260
150

FIG._8

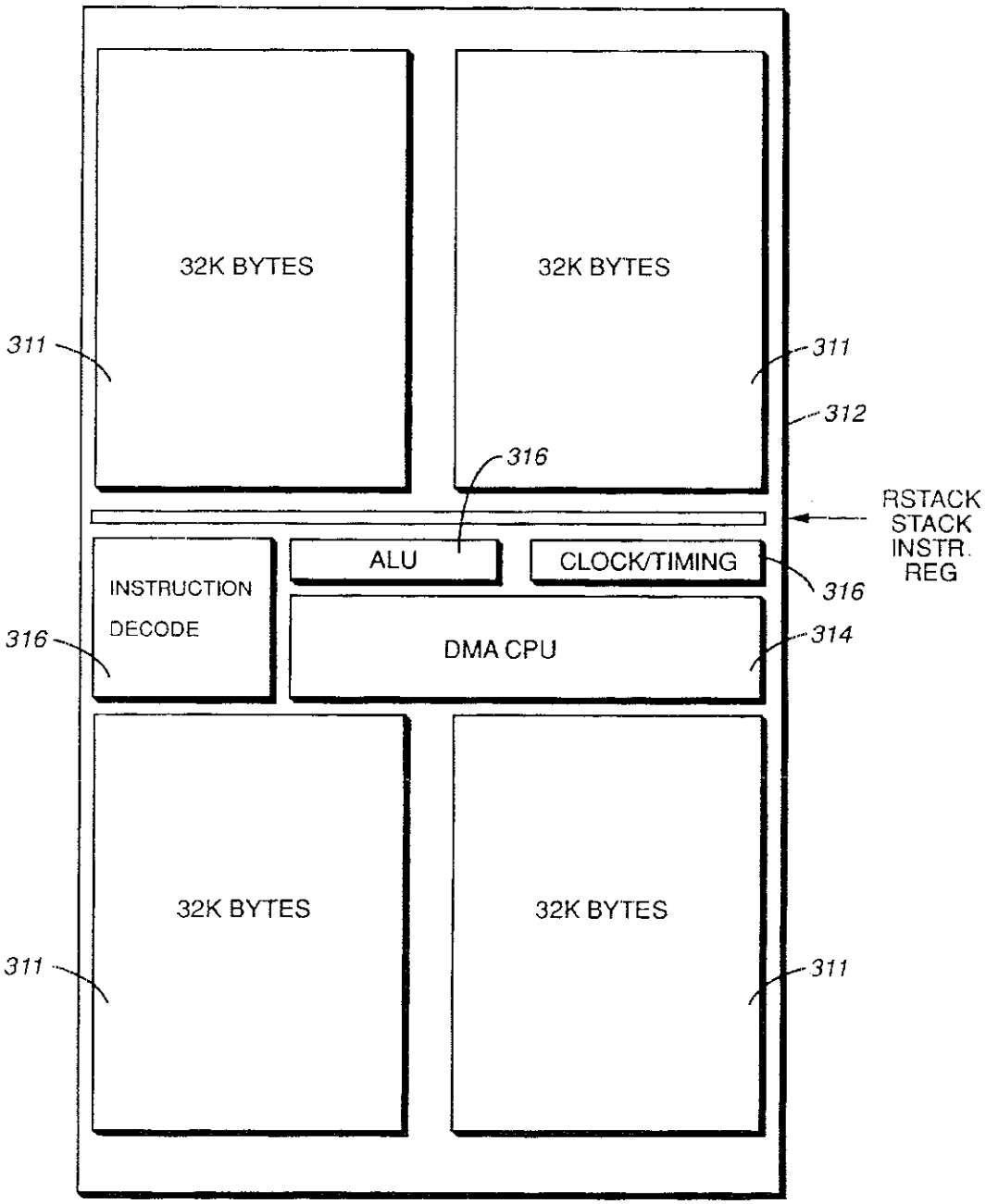

32K BYTES
32K BYTES
311
311
312
316
RSTACK
STACK
INSTR.
REG
ALU
CLOCK/TIMING
316
INSTRUCTION
DECODE
316
DMA CPU
314
32K BYTES
32K BYTES
311
311

FIG._9

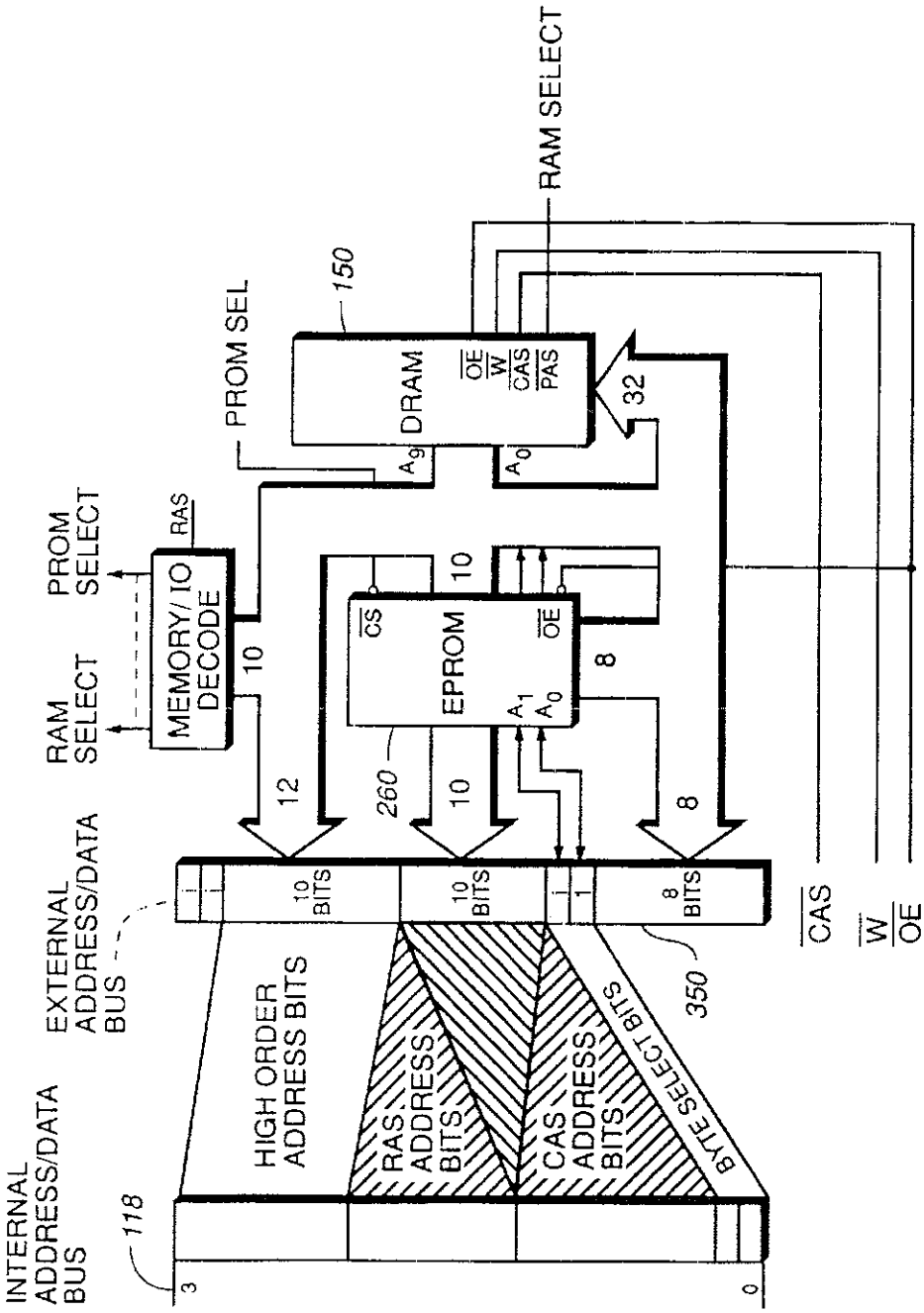
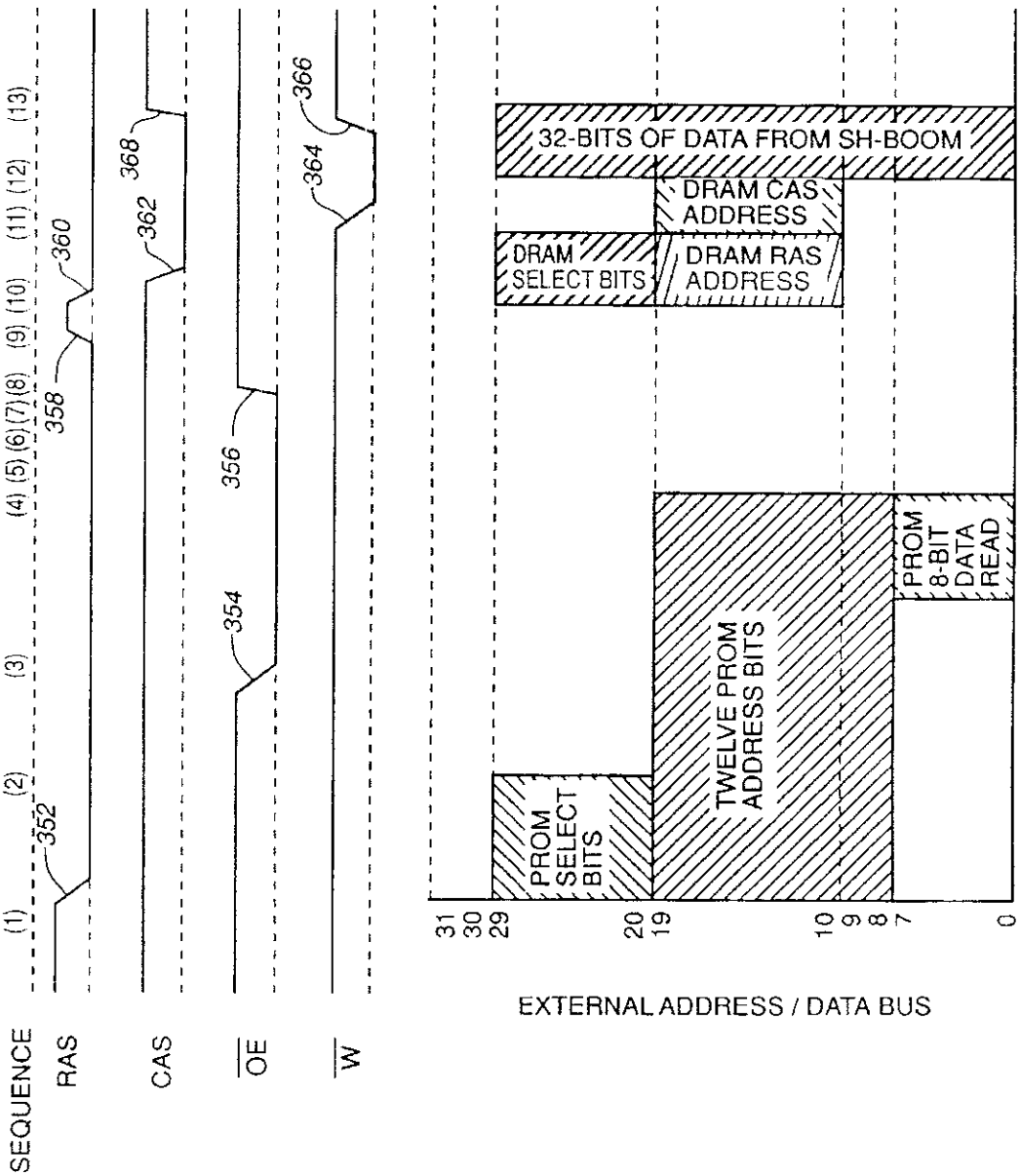

RAM SELECT
PROM SEL
150
DRAM
OE
W
CAS
RAS
32
A9
A0
PROM SELECT
RAS
MEMORY/IO DECODE
10
RAM SELECT
CS
EPROM
OE
10
8
A1
A0
260
12
10
8
EXTERNAL ADDRESS/DATA BUS
10 BITS
10 BITS
1
1
8 BITS
CAS
W
OE
350
HIGH ORDER ADDRESS BITS
RAS ADDRESS BITS
CAS ADDRESS BITS
BYTE SELECT BITS
INTERNAL ADDRESS/DATA BUS
118
3
0

FIG._10

FIG._11


SEQUENCE
RAS
CAS
OE
W
(1)
(2)
(3)
(4) (5) (6) (7) (8)
(9) (10)
(11) (12)
(13)
352
354
356
358
360
362
364
366
368
32-BITS OF DATA FROM SH-BOOM
DRAM CAS ADDRESS
DRAM SELECT BITS
DRAM RAS ADDRESS
PROM SELECT BITS
TWELVE PROM ADDRESS BITS
PROM 8-BIT DATA READ
31 30 29 20 19 10 9 8 7 0
EXTERNAL ADDRESS / DATA BUS

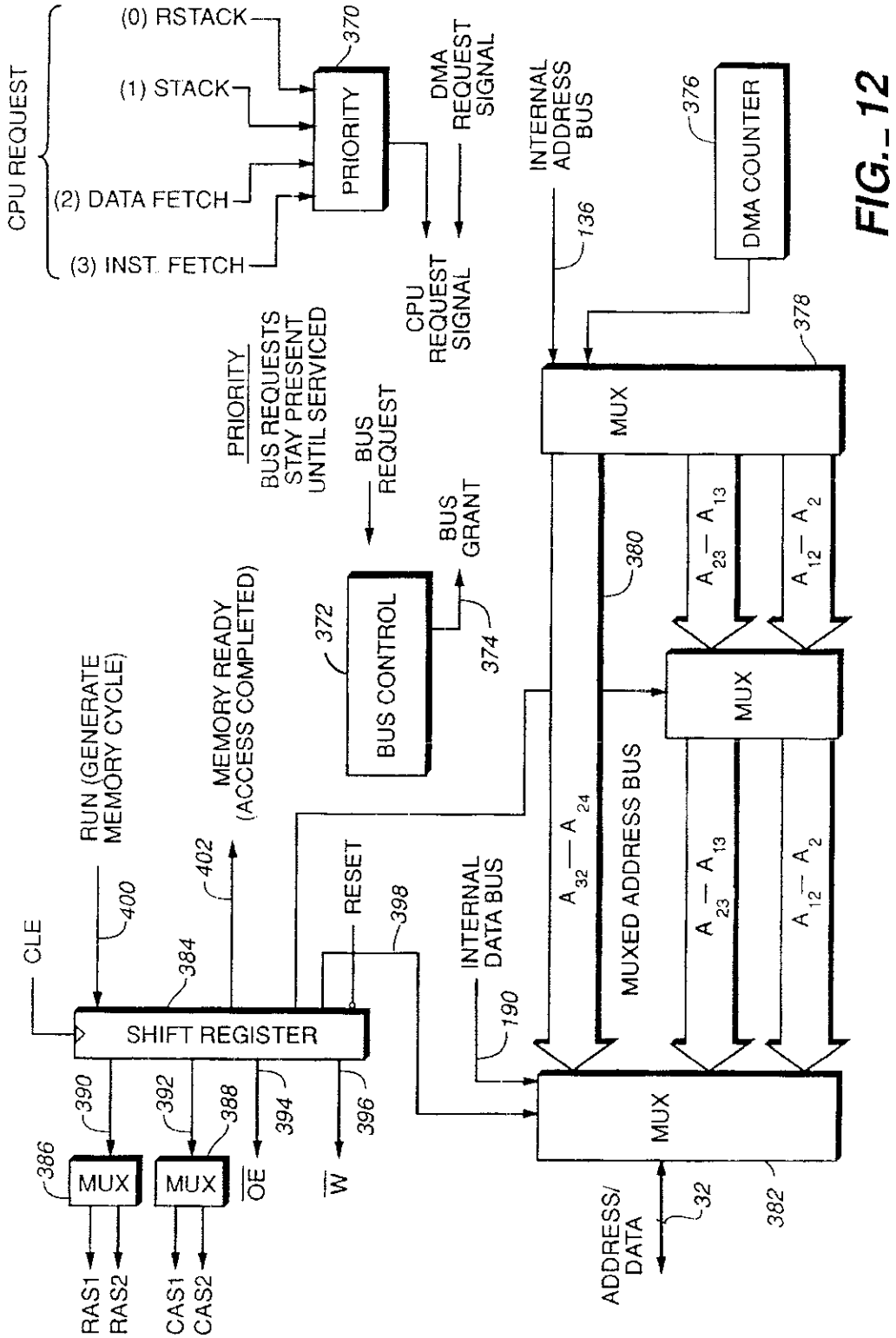

CPU REQUEST
(0) RSTACK
(1) STACK
(2) DATA FETCH
(3) INST. FETCH
370
PRIORITY
DMA REQUEST SIGNAL
CPU REQUEST SIGNAL
INTERNAL ADDRESS BUS
136
376
DMA COUNTER
PRIORITY
BUS REQUESTS STAY PRESENT UNTIL SERVICED
BUS REQUEST
BUS GRANT
372
BUS CONTROL
374
MEMORY READY (ACCESS COMPLETED)
RUN (GENERATE MEMORY CYCLE)
400
402
CLE
384
SHIFT REGISTER
RESET
398
INTERNAL DATA BUS
190
MUXED ADDRESS BUS
378
MUX
380
A_{23} — A_{13}
A_{12} — A_2
A_{32} — A_{24}
386
390
392
388
394
396
MUX
MUX
$\overline{OE}$
$\overline{W}$
RAS1
RAS2
CAS1
CAS2
ADDRESS/ DATA
32
382

FIG._12

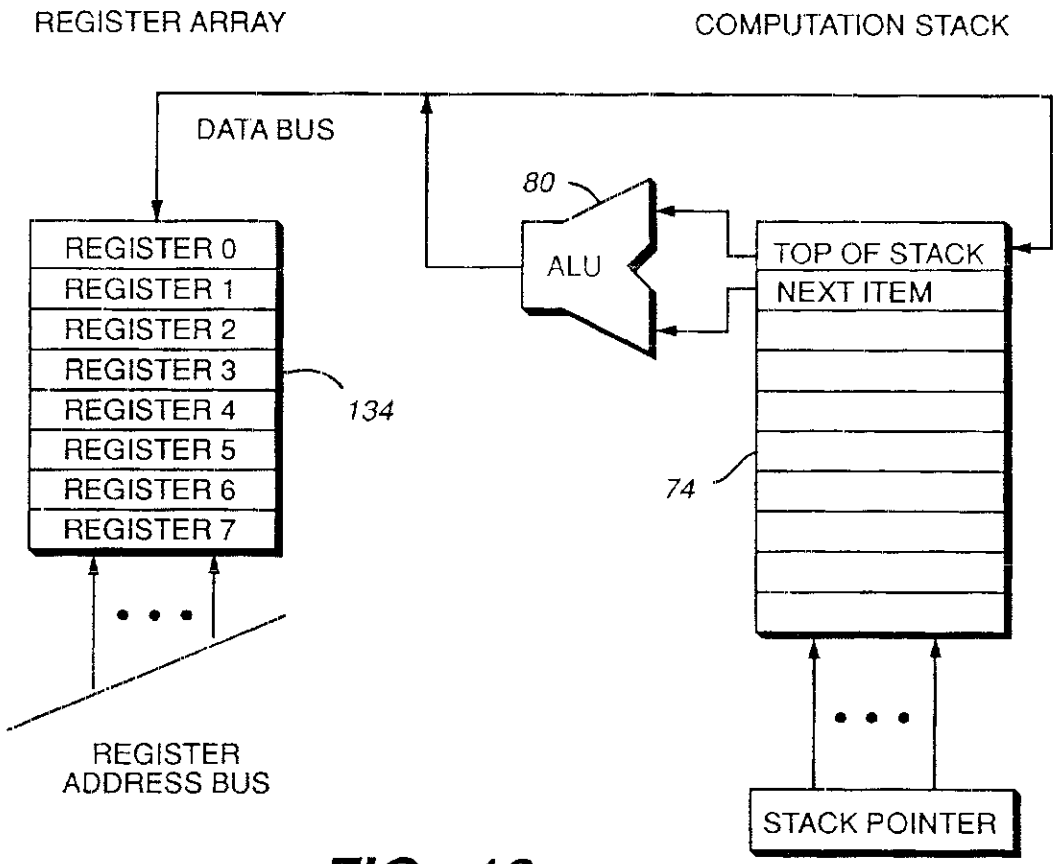

REGISTER ARRAY
COMPUTATION STACK
DATA BUS
80
ALU
REGISTER 0
REGISTER 1
REGISTER 2
REGISTER 3
REGISTER 4
REGISTER 5
REGISTER 6
REGISTER 7
134
TOP OF STACK
NEXT ITEM
74
REGISTER
ADDRESS BUS
STACK POINTER

FIG._13

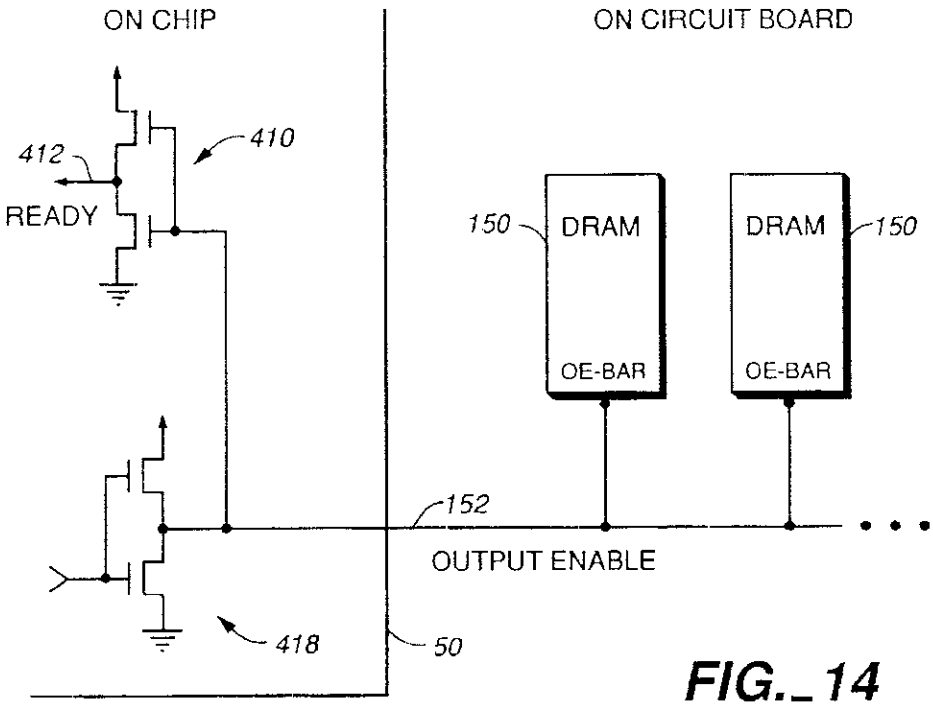

ON CHIP
ON CIRCUIT BOARD
412
410
READY
150
DRAM
OE-BAR
DRAM
OE-BAR
150
152
OUTPUT ENABLE
418
50

FIG._14

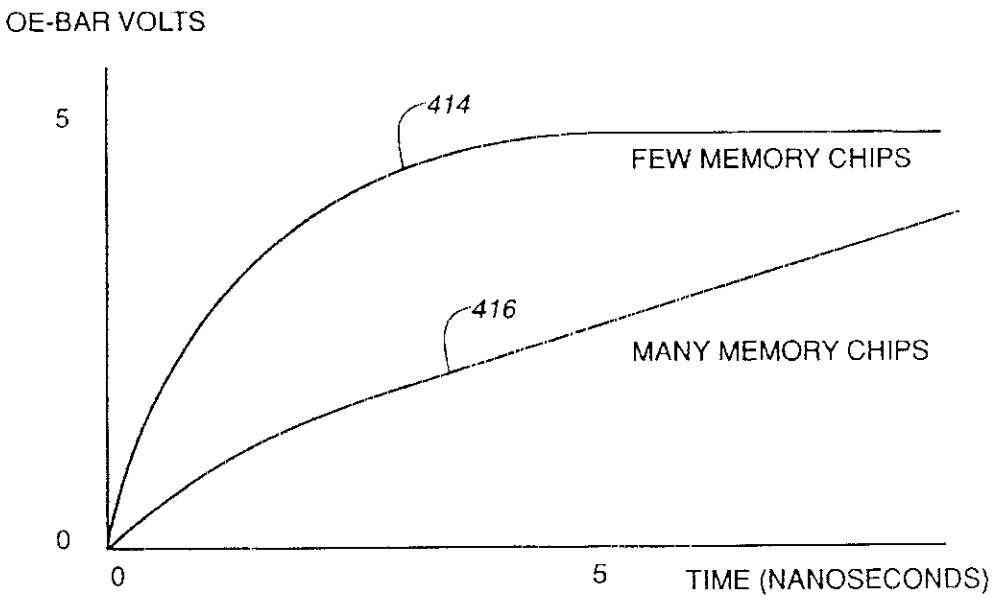

OE-BAR VOLTS
5
0
414
FEW MEMORY CHIPS
416
MANY MEMORY CHIPS
0
5
TIME (NANOSECONDS)

FIG._15

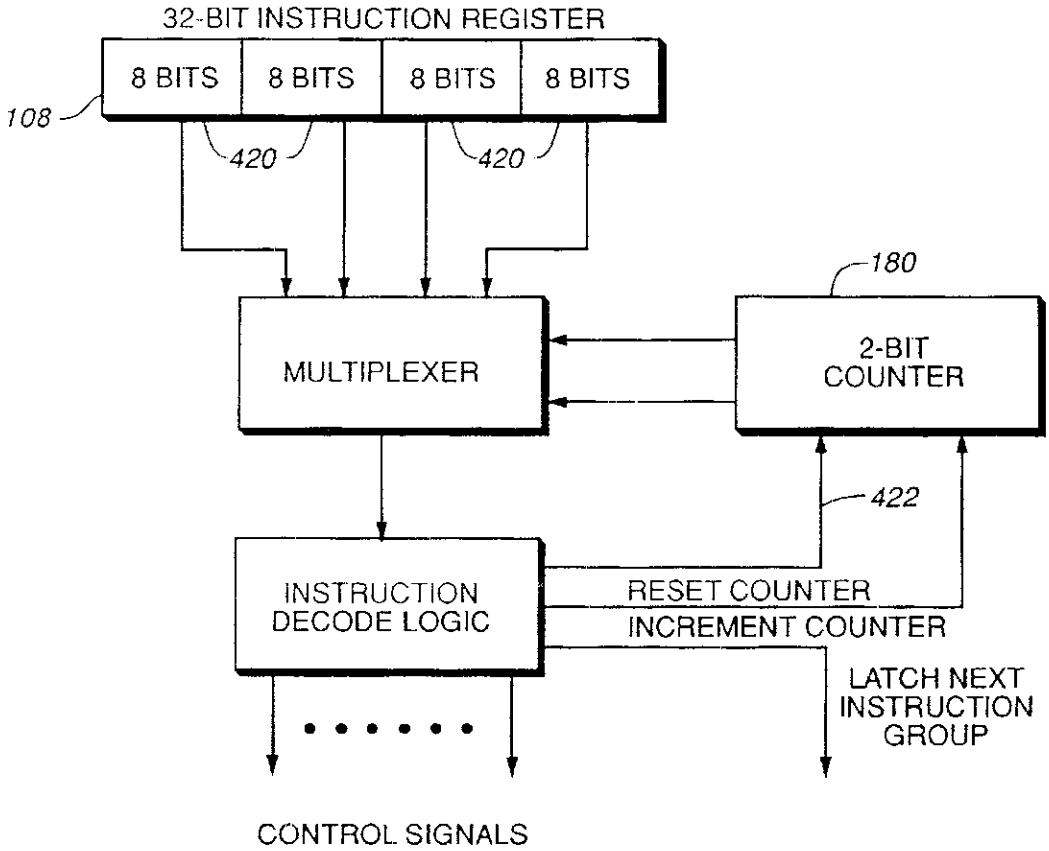

32-BIT INSTRUCTION REGISTER
8 BITS
8 BITS
8 BITS
8 BITS
108
420
420
180
MULTIPLEXER
2-BIT COUNTER
422
INSTRUCTION DECODE LOGIC
RESET COUNTER
INCREMENT COUNTER
LATCH NEXT INSTRUCTION GROUP
CONTROL SIGNALS

FIG._16

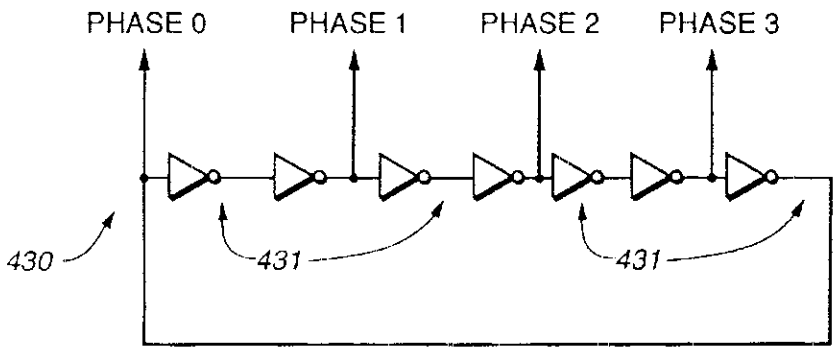

PHASE 0
PHASE 1
PHASE 2
PHASE 3
430
431
431

FIG._18

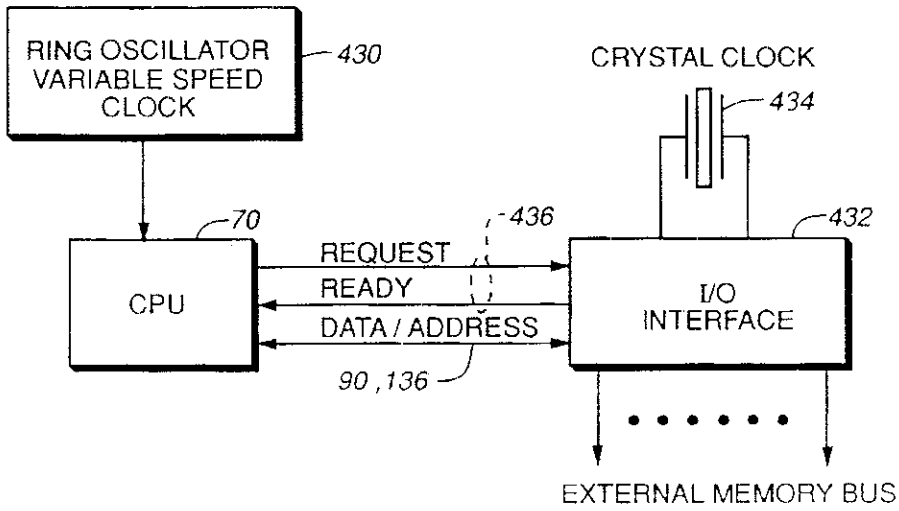

RING OSCILLATOR
VARIABLE SPEED
CLOCK
430
CRYSTAL CLOCK
434
70
436
432
CPU
REQUEST
READY
DATA / ADDRESS
I/O
INTERFACE
90 , 136
EXTERNAL MEMORY BUS

FIG._17

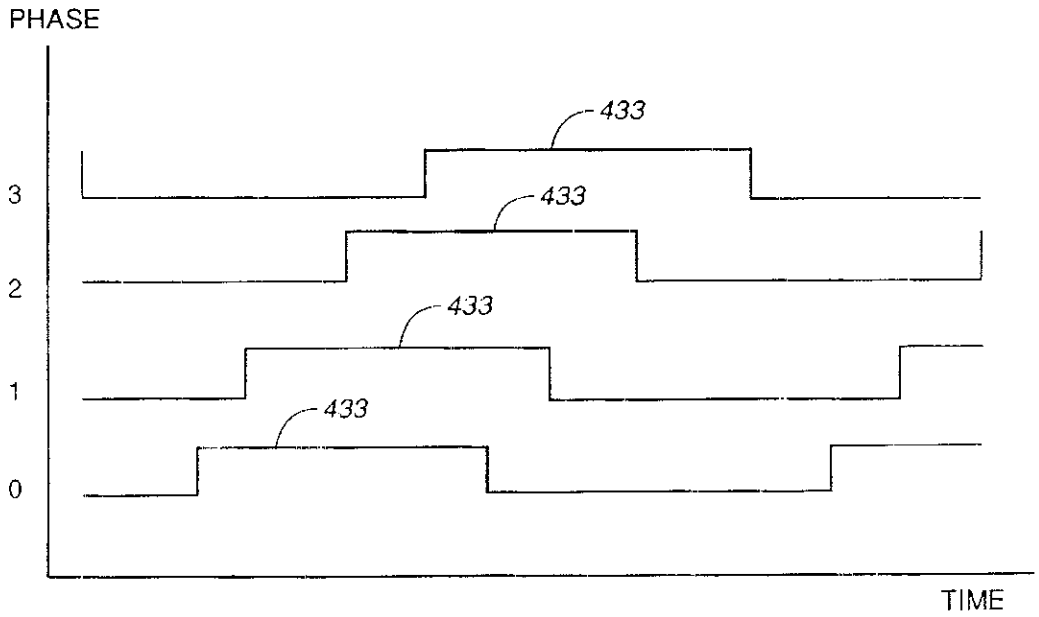

PHASE
433
433
433
433
3
2
1
0
TIME

FIG._19

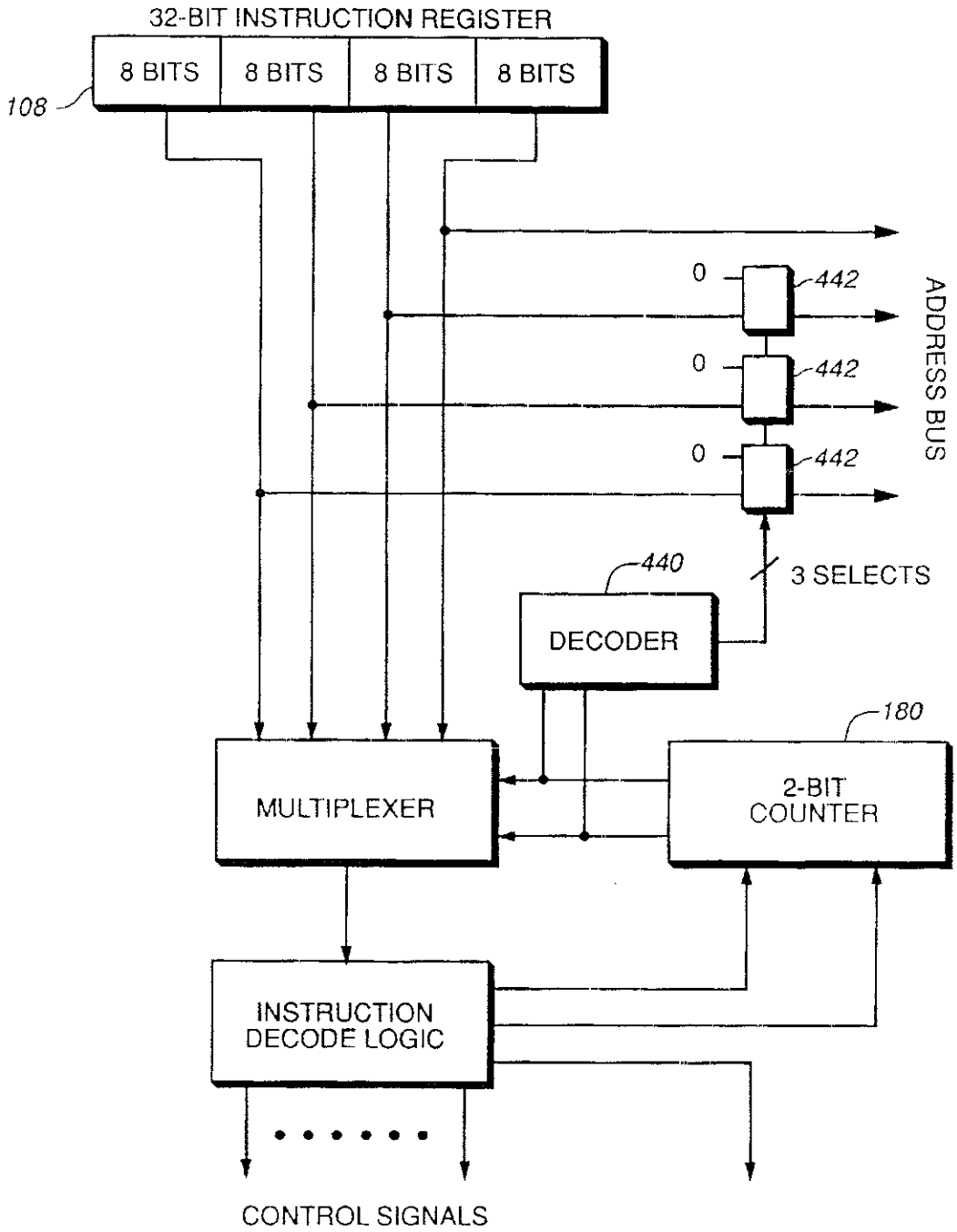

32-BIT INSTRUCTION REGISTER
8 BITS
8 BITS
8 BITS
8 BITS
108
0
442
0
442
0
442
ADDRESS BUS
440
3 SELECTS
DECODER
180
MULTIPLEXER
2-BIT COUNTER
INSTRUCTION DECODE LOGIC
CONTROL SIGNALS

FIG._20

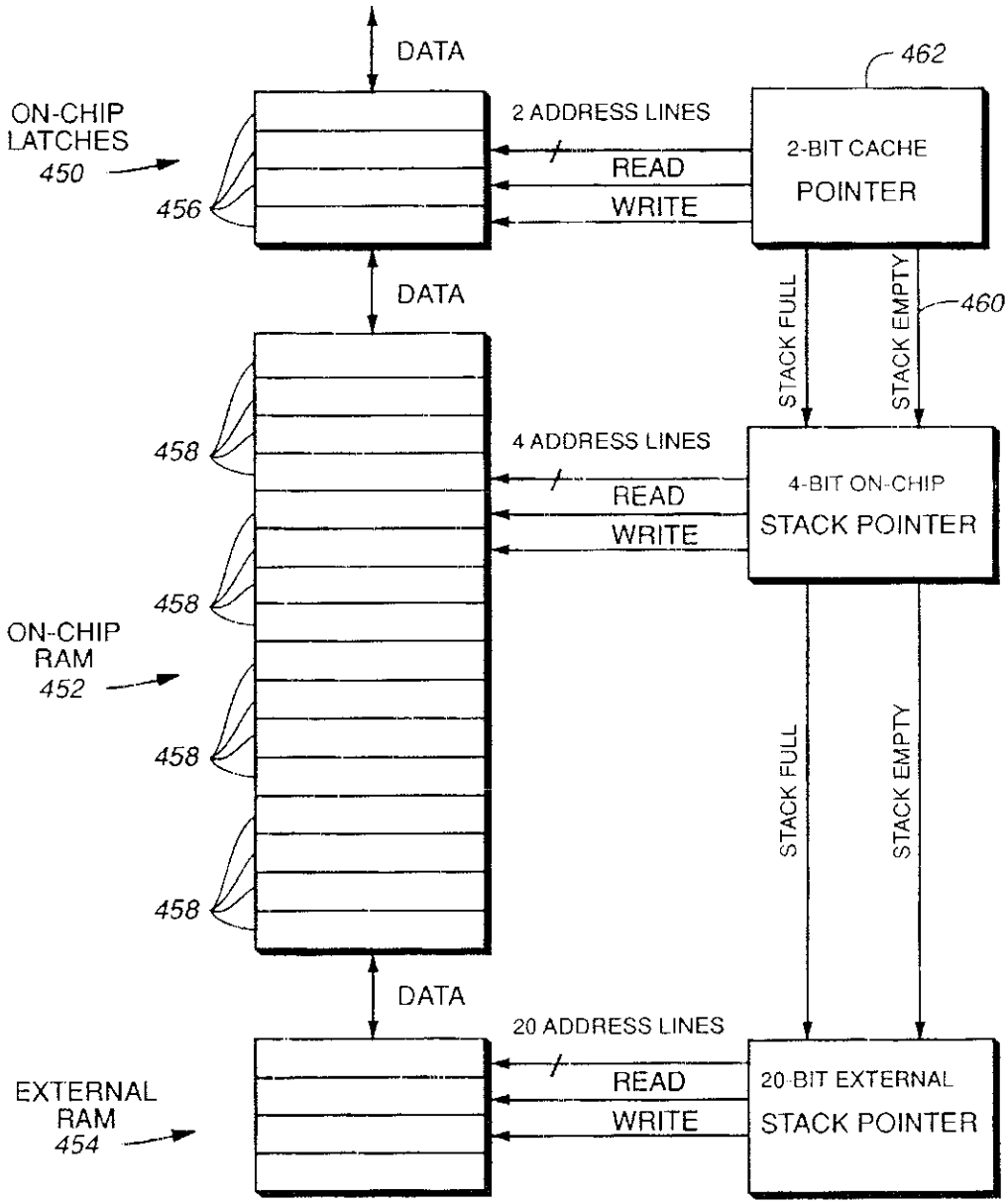

DATA
462
ON-CHIP
LATCHES
450
2 ADDRESS LINES
READ
WRITE
2-BIT CACHE
POINTER
456
DATA
STACK FULL
STACK EMPTY
460
458
4 ADDRESS LINES
READ
WRITE
4-BIT ON-CHIP
STACK POINTER
458
ON-CHIP
RAM
452
458
458
STACK FULL
STACK EMPTY
DATA
20 ADDRESS LINES
READ
WRITE
20-BIT EXTERNAL
STACK POINTER
EXTERNAL
RAM
454

FIG._21

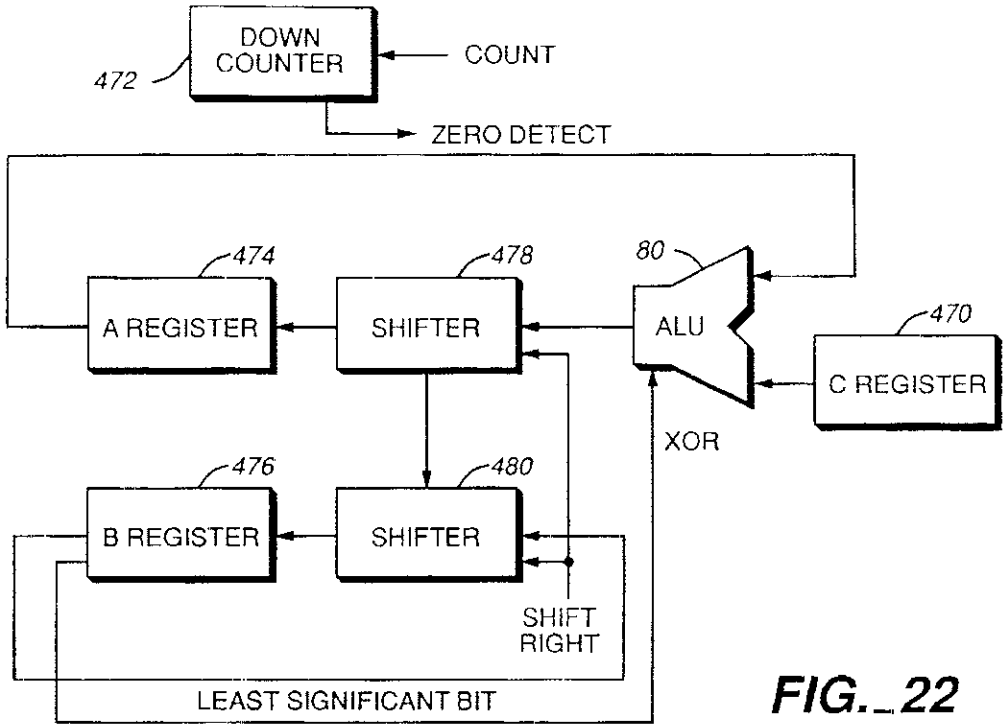

DOWN COUNTER
472
COUNT
ZERO DETECT
474
A REGISTER
478
SHIFTER
80
ALU
470
C REGISTER
XOR
476
B REGISTER
480
SHIFTER
SHIFT RIGHT
LEAST SIGNIFICANT BIT

FIG._22

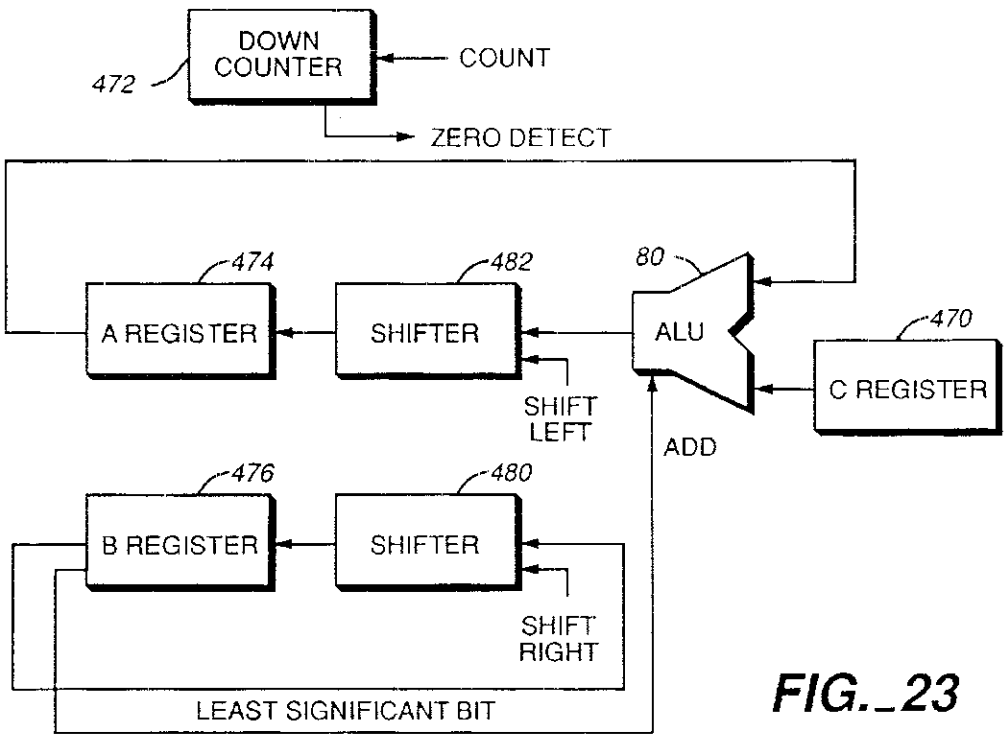

DOWN COUNTER
472
COUNT
ZERO DETECT
474
A REGISTER
482
SHIFTER
80
ALU
470
C REGISTER
SHIFT LEFT
ADD
476
B REGISTER
480
SHIFTER
SHIFT RIGHT
LEAST SIGNIFICANT BIT

FIG._23

HIGH PERFORMANCE MICROPROCESSOR HAVING VARIABLE SPEED SYSTEM CLOCK

CROSS REFERENCE TO RELATED APPLICATIONS

This application is a division of U S. application Ser No 07/389,334 filed Aug 3 1989 now U S Pat No 5,440,749

BACKGROUND OF THE INVENTION

1 Field of the Invention

The present invention relates generally to a simplified reduced instruction set computer (RISC) microprocessor More particularly it relates to such a microprocessor which is capable of performance levels of, for example, 20 million instructions per second (MIPS) at a price of, for example, 20 dollars

2 Description of the Prior Art

Since the invention of the microprocessor improvements in its design have taken two different approaches In the first approach a brute force gain in performance has been achieved through the provision of greater numbers of faster transistors in the microprocessor integrated circuit and an instruction set of increased complexity. This approach is exemplified by the Motorola 68000 and Intel 80X86 microprocessor families The trend in this approach is to larger die sizes and packages with hundreds of pinouts

More recently it has been perceived that performance gains can be achieved through comparative simplicity, both in the microprocessor integrated circuit itself and in its instruction set This second approach provides RISC microprocessors, and is exemplified by the Sun SPARC and the Intel 8960 microprocessors However, even with this approach as conventionally practiced, the packages for the microprocessor are large in order to accommodate the large number of pinouts that continue to be employed. A need therefore remains for further simplification of high performance microprocessors

With conventional high performance microprocessors fast static memories are required for direct connection to the microprocessors in order to allow memory accesses that are fast enough to keep up with the microprocessors Slower dynamic random access memories (DRAMs) are used with such microprocessors only in a hierarchical memory arrangement with the static memories acting as a buffer between the microprocessors and the DRAMs The necessity to use static memories increases cost of the resulting systems

Conventional microprocessors provide direct memory accesses (DMA) for system peripheral units through DMA controllers which may be located on the microprocessor integrated circuit or provided separately Such DMA controllers can provide routine handling of DMA requests and responses, but some processing by the main central processing unit (CPU) of the microprocessor is required

SUMMARY OF THE INVENTION

Accordingly, it is an object of this invention to provide a microprocessor with a reduced pin count and cost compared to conventional microprocessors

It is another object of the invention to provide a high performance microprocessor that can be directly connected to DRAMs without sacrificing microprocessor speed

It is a further object of the invention to provide a high performance microprocessor in which DMA does not require use of the main CPU during DMA requests and responses and which provides very rapid DMA response with predictable response times

The attainment of these and related objects may be achieved through use of the novel high performance, low cost microprocessor herein disclosed In accordance with one aspect of the invention a microprocessor system in accordance with this invention has a central processing unit, a dynamic random access memory and a bus connecting the central processing unit to the dynamic random access memory There is a multiplexing means on the bus between the central processing unit and the dynamic random access memory The multiplexing means is connected and configured to provide row addresses column addresses and data on the bus

In accordance with another aspect of the invention, the microprocessor system has a means connected to the bus for fetching instructions for the central processing unit on the bus. The means for fetching instructions is configured to fetch multiple sequential instructions in a single memory cycle In a variation of this aspect of the invention, a programmable read only memory containing instructions for the central processing unit is connected to the bus The means for fetching instructions includes means for assembling a plurality of instructions from the programmable read only memory and storing the plurality of instructions in the dynamic random access memory

In another aspect of the invention, the microprocessor system includes a central processing unit a direct memory access processing unit and a memory connected by a bus The direct memory access processing unit includes means for fetching instructions for the central processing unit and for fetching instructions for the direct memory access processing unit on the bus

In a further aspect of the invention, the microprocessor system, including the memory, is contained in an integrated circuit The memory is a dynamic random access memory and the means for fetching multiple instructions includes a column latch for receiving the multiple instructions

In still another aspect of the invention, the microprocessor system additionally includes an instruction register for the multiple instructions connected to the means for fetching instructions A means is connected to the instruction register for supplying the multiple instructions in succession from the instruction register. A counter is connected to control the means for supplying the multiple instructions to supply the multiple instructions in succession A means for decoding the multiple instructions is connected to receive the multiple instructions in succession from the means for supplying the multiple instructions. The counter is connected to said means for decoding to receive incrementing and reset control signals from the means for decoding The means for decoding is configured to supply the reset control signal to the counter and to supply a control signal to the means for fetching instructions in response to a SKIP instruction in the multiple instructions In a modification of this aspect of the invention, the microprocessor system additionally has a loop counter connected to receive a decrement control signal from the means for decoding The means for decoding is configured to supply the reset control signal to the counter and the decrement control signal to the loop counter in response to a MICROLOOP instruction in the multiple instructions. In a further modification to this aspect of the invention, the means for decoding is configured to control

the counter in response to an instruction utilizing a variable width operand A means is connected to the counter to select the variable width operand in response to the counter

In a still further aspect of the invention the microprocessor system includes an arithmetic logic unit A first push down stack is connected to the arithmetic logic unit The first push down stack includes means for storing a top item connected to a first input of the arithmetic logic unit and means for storing a next item connected to a second input of the arithmetic logic unit The arithmetic logic unit has an output connected to the means for storing a top item The means for storing a top item is connected to provide an input to a register file The register file desirably is a second push down stack and the means for storing a top item and the register file are bidirectionally connected

In another aspect of the invention, a data processing system has a microprocessor including a sensing circuit and a driver circuit a memory and an output enable line connected between the memory, the sensing circuit and the driver circuit The sensing circuit is configured to provide a ready signal when the output enable line reaches a predetermined electrical level, such as a voltage The microprocessor is configured so that the driver circuit provides an enabling signal on the output enable line responsive to the ready signal

In a further aspect of the invention the microprocessor system has a ring counter variable speed system clock connected to the central processing unit. The central processing unit and the ring counter variable speed system clock are provided in a single integrated circuit. An input/output interface is connected to exchange coupling control signals, addresses and data with the input/output interface A second clock independent of the ring counter variable speed system clock is connected to the input/output interface

In yet another aspect of the invention, a push down stack is connected to the arithmetic logic unit The push down stack includes means for storing a top item connected to a first input of the arithmetic logic unit and means for storing a next item connected to a second input of the arithmetic logic unit The arithmetic logic unit has an output connected to the means for storing a top item The push down stack has a first plurality of stack elements configured as latches and a second plurality of stack elements configured as a random access memory The first and second plurality of stack elements and the central processing unit are provided in a single integrated circuit A third plurality of stack elements is configured as a random access memory external to the single integrated circuit In this aspect of the invention, desirably a first pointer is connected to the first plurality of stack elements, a second pointer connected to the second plurality of stack elements, and a third pointer is connected to the third plurality of stack elements The central processing unit is connected to pop items from the first plurality of stack elements The first stack pointer is connected to the second stack pointer to pop a first plurality of items from the second plurality of stack elements when the first plurality of stack elements are empty from successive pop operations by the central processing unit The second stack pointer is connected to the third stack pointer to pop a second plurality of items from the third plurality of stack elements when the second plurality of stack elements are empty from successive pop operations by the central processing unit

In another aspect of the invention a first register is connected to supply a first input to the arithmetic logic unit A first shifter is connected between an output of the arithmetic logic unit and the first register. A second register is connected to receive a starting polynomial value An output of the second register is connected to a second shifter A least significant bit of the second register is connected to The arithmetic logic unit. A third register is connected to supply feedback terms of a polynomial to the arithmetic logic unit A down counter, for counting down a number corresponding to digits of a polynomial to be generated, is connected to the arithmetic logic unit The arithmetic logic unit is responsive to a polynomial instruction to carry out an exclusive OR of the contents of the first register with the contents of the third register if the least significant bit of the second register is a "ONE" and to pass the contents of the first register unaltered if the least significant bit of the second register is a "ZERO", until the down counter completes a count. The polynomial to be generated results in said first register

In still another aspect of the invention, a result register is connected to supply a first input to the arithmetic logic unit A first, left shifting shifter is connected between an output of the arithmetic logic unit and the result register A multiplier register is connected to receive a multiplier in bit reversed form An output of the multiplier register is connected to a second, right shifting shifter A least significant bit of the multiplier register is connected to the arithmetic logic unit. A third register is connected to supply a multiplicand to said arithmetic logic unit A down counter for counting down a number corresponding to one less than the number of digits of the multiplier, is connected to the arithmetic logic unit The arithmetic logic unit is responsive to a multiply instruction to add the contents of the result register with the contents of the third register, when the least significant bit of the multiplier register is a "ONE" and to pass the contents of the result register unaltered, until the down counter completes a count The product results in the result register

The attainment of the foregoing and related objects, advantages and features of the invention should be more readily apparent to those skilled in the art, after review of the following more detailed description of the invention taken together with the drawings in which:

BRIEF DESCRIPTION OF THE DRAWINGS

FIG 1 is an external plan view of an integrated circuit package incorporating a microprocessor in accordance with the invention

FIG 2 is a block diagram of a microprocessor in accordance with the invention

FIG 3 is a block diagram of a portion of a data processing system incorporating the microprocessor of FIGS 1 and 2

FIG 4 is a more detailed block diagram of a portion of the microprocessor shown in FIG 2

FIG 5 is a more detailed block diagram of another portion of the microprocessor shown in FIG 2

FIG 6 is a block diagram of another portion of the data processing system shown in part in FIG. 3 and incorporating the microprocessor of FIGS 1–2 and 4–5

FIGS 7 and 8 are layout diagrams for the data processing system shown in part in FIGS 3 and 6.

FIG 9 is a layout diagram of a second embodiment of a microprocessor in accordance with the invention in a data processing system on a single integrated circuit

FIG 10 is a more detailed block diagram of a portion of the data processing system of FIGS 7 and 8

FIG 11 is a timing diagram useful for understanding operation of the system portion shown in FIG 12

FIG 12 is another more detailed block diagram of a further portion of the data processing system of FIGS 7 and 8

FIG. 13 is a more detailed block diagram of a portion of the microprocessor shown in FIG. 2.

FIG. 14 is a more detailed block and schematic diagram of a portion of the system shown in FIGS. 3 and 7–8.

FIG. 15 is a graph useful for understanding operation of the system portion shown in FIG. 14.

FIG. 16 is a more detailed block diagram showing part of the system portion shown in FIG. 4.

FIG. 17 is a more detailed block diagram of a portion of the microprocessor shown in FIG. 2.

FIG. 18 is a more detailed block diagram of part of the microprocessor portion shown in FIG. 17.

FIG. 19 is a set of waveform diagrams useful for understanding operation of the part of the microprocessor portion shown in FIG. 18.

FIG. 20 is a more detailed block diagram showing another part of the system portion shown in FIG. 4.

FIG. 21 is a more detailed block diagram showing another part of the system portion shown in FIG. 4.

FIGS. 22 and 23 are more detailed block diagrams showing another part of the system portion shown in FIG. 4.

DETAILED DESCRIPTION OF THE INVENTION

Overveiw

The microprocessor of this invention is desirably implemented as a 32-bit microprocessor optimized for:

HIGH EXECUTION SPEED, and

LOW SYSTEM COST

In this embodiment, the microprocessor can be thought of as 20 MIPS for 20 dollars. Important distinguishing features of the microprocessor are:

Uses low-cost commodity DYNAMIC RAMS to run 20 MIPS

4 instruction fetch per memory cycle

On-chip fast page-mode memory management

Runs fast without external cache

Requires few interfacing chips

Crams 32-bit CPU in 44 pin SOJ package

The instruction set is organized so that most operations can be specified with 8-bit instructions. Two positive products of this philosophy are:

Programs are smaller.

Programs can execute much faster

The bottleneck in most computer systems is the memory bus. The bus is used to fetch instructions and fetch and store data. The ability to fetch four instructions in a single memory bus cycle significantly increases the bus availability to handle data.

Turning now to the drawings, more particularly to FIG. 1, there is shown a packaged 32-bit microprocessor **50** in a 44-pin plastic leadless chip carrier, shown approximately 100 times its actual size of about 0.8 inch on a side. The fact that the microprocessor **50** is provided as a 44-pin package represents a substantial departure from typical microprocessor packages, which usually have about 200 input/output (I/O) pins. The microprocessor **50** is rated at 20 million instructions per second (MIPS). Address and data lines **52**, also labelled D0–D31, are shared for addresses and data without speed penalty as a result of the manner in which the microprocessor **50** operates, as will be explained below.

DYNAMIC RAM

In addition to the low cost 44-pin package, another unusual aspect of the high performance microprocessor **50** is that it operates directly with dynamic random access memories (DRAMs), as shown by row address strobe (RAS) and column address strobe (CAS) I/O pins **54**. The other I/O pins for the microprocessor **50** include V_{DD} pins **56**, V_{SS} pins **58**, output enable pin **60**, write pin **62**, clock pin **64** and reset pin **66**.

All high speed computers require high speed and expensive memory to keep up. The highest speed static RAM memories cost as much as ten times as much as slower dynamic RAMs. This microprocessor has been optimized to use low-cost dynamic RAM in high-speed page-mode. Page-mode dynamic RAMs offer static RAM performance without the cost penalty. For example, low-cost 85 nsec dynamic RAMs access at 25 nsec when operated in fast page-mode. Integrated fast page-mode control on the microprocessor chip simplifies system interfacing and results in a faster system.

Details of the microprocessor **50** are shown in FIG. 2. The microprocessor **50** includes a main central processing unit (CPU) **70** and a separate direct memory access (DMA) CPU **72** in a single integrated circuit making up the microprocessor **50**. The main CPU **70** has a first 16 deep push down stack **74**, which has a top item register **76** and a next item register **78**, respectively connected to provide inputs to an arithmetic logic unit (ALU) **80** by lines **82** and **84**. An output of the ALU **80** is connected to the top item register **76** by line **86**. The output of the top item register at 82 is also connected by line **88** to an internal data bus **90**.

A loop counter **92** is connected to a decrementer **94** by lines **96** and **98**. The loop counter **92** is bidirectionally connected to the internal data bus **90** by line **100**. Stack pointer **102**, return stack pointer **104**, mode register **106** and instruction register **108** are also connected to the internal data bus **90** by lines **110**, **112**, **114** and **116**, respectively. The internal data bus **90** is connected to memory controller **118** and to gate **120**. The gate **120** provides inputs on lines **122**, **124**, and **126** to X register **128**, program counter **130** and Y register **132** of return push down stack **134**. The X register **128**, program counter **130** and Y register **132** provide outputs to internal address bus **136** on lines **138**, **140** and **142**. The internal address bus provides inputs to the memory controller **118** and to an incrementer **144**. The incrementer **144** provides inputs to the X register, program counter and Y register via lines **146**, **122**, **124** and **126**. The DMA CPU **72** provides inputs to the memory controller **118** on line **148**. The memory controller **118** is connected to a RAM (not shown) by address/data bus **150** and control lines **152**.

FIG. 2 shows that the microprocessor **50** has a simple architecture. Prior art RISC microprocessors are substantially more complex in design. For example, the SPARC RISC microprocessor has three times the gates of the microprocessor **50**, and the Intel 8960 RISC microprocessor has 20 times the gates of the microprocessor **50**. The speed of this microprocessor is in substantial part due to this simplicity. The architecture incorporates push down stacks and register write to achieve this simplicity.

The microprocessor **50** incorporates an I/O that has been tuned to make heavy use of resources provided on the integrated circuit chip. On chip latches allow use of the same I/O circuits to handle three different things: column addressing, row addressing and data, with a slight to non-existent speed penalty. This triple bus multiplexing results in fewer buffers to expand, fewer interconnection lines, fewer I/O pins and fewer internal buffers.

The provision of on-chip DRAM control gives a performance equal to that obtained with the use of static RAMs. As a result, memory is provided at ¼ the system cost of static RAM used in most RISC systems.

The microprocessor **50** fetches 4 instructions per memory cycle; the instructions are in an 8-bit format, and this is a 32-bit microprocessor. System speed is therefore 4 times the memory bus bandwidth. This ability enables the microprocessor to break the Von Neumann bottleneck of the speed of getting the next instruction. This mode of operation is possible because of the use of a push down stack and register array. The push down stack allows the use of implied addresses, rather than the prior art technique of explicit addresses for two sources and a destination.

Most instructions execute in 20 nanoseconds in the microprocessor **50**. The microprocessor can therefore execute instructions at 50 peak MIPS without pipeline delays. This is a function of the small number of gates in the microprocessor **50** and the high degree of parallelism in the architecture of the microprocessor.

FIG. **3** shows how column and row addresses are multiplexed on lines D8–D14 of the microprocessor **50** for addressing DRAM **150** from I/O pins **52**. The DRAM **150** is one of eight, but only one DRAM **150** has been shown for clarity. As shown, the lines D11–D18 are respectively connected to row address inputs A0–A8 of the DRAM **150**. Additionally, lines D12–D15 are connected to the data inputs DQ1–DQ4 of the DRAM **150**. The output enable, write and column address strobe pins **54** are respectively connected to the output enable, write and column address strobe inputs of the DRAM **150** by lines **152**. The row address strobe pin **54** is connected through row address strobe decode logic **154** to the row address strobe input of the DRAM **150** by lines **156** and **158**.

D0–D7 pins **52** (FIG. **1**) are idle when the microprocessor **50** is outputting multiplexed row and column addresses on D11–D18 pins **52**. The D0–D7 pins **52** can therefore simultaneously be used for I/O when right justified I/O is desired. Simultaneous addressing and I/O can therefore be carried out.

FIG. **4** shows how the microprocessor **50** is able to achieve performance equal to the use of static RAMS with DRAMs through multiple instruction fetch in a single clock cycle and instruction fetch-ahead. Instruction register **108** receives four 8-bit byte instruction words 1–4 on 32-bit internal data bus **90**. The four instruction byte 1–4 locations of the instruction register **108** are connected to multiplexer **170** by busses **172**, **174**, **176** and **178**, respectively. A microprogram counter **180** is connected to the multiplexer **170** by lines **182**. The multiplexer **170** is connected to decoder **184** by bus **186**. The decoder **184** provides internal signals to the rest of the microprocessor **50** on lines **188**.

Most significant bits **190** of each instruction byte 1–4 location are connected to a 4-input decoder **192** by lines **194**. The output of decoder **192** is connected to memory controller **118** by line **196**. Program counter **130** is connected to memory controller **118** by internal address bus **136** and the instruction register **108** is connected to the memory controller **118** by the internal data bus **90**. Address/data bus **198** and control bus **200** are connected to the DRAMS **150** (FIG. **3**).

In operation, when the most significant bits **190** of remaining instructions 1–4 are "1" in a clock cycle of the microprocessor **50**, there are no memory reference instructions in the queue. The output of decoder **192** on line **196** requests an instruction fetch ahead by memory controller **118** without interference with other accesses. While the current instructions in instruction register **108** are executing, the memory controller **118** obtains the address of the next set of four instructions from program counter **130** and obtains that set of instructions. By the time the current set of instructions has completed execution, the next set of instructions is ready for loading into the instruction register.

Details of the DMA CPU **72** are provided in FIG. **5**. Internal data bus **90** is connected to memory controller **118** and to DMA instruction register **210**. The DMA instruction register **210** is connected to DMA program counter **212** by bus **214**, to transfer size counter **216** by bus **218** and to timed transfer interval counter **220** by bus **222**. The DMA instruction register **210** is also connected to DMA I/O and RAM address register **224** by line **226**. The DMA I/O and RAM address register **224** is connected to the memory controller **118** by memory cycle request line **228** and bus **230**. The DMA program counter **212** is connected to the internal address bus **136** by bus **232**. The transfer size counter **216** is connected to a DMA instruction done decrementer **234** by lines **236** and **238**. The decrementer **234** receives a control input on memory cycle acknowledge line **240**. When transfer size counter **216** has completed its count, it provides a control signal to DMA program counter **212** on line **242**. Timed transfer interval counter **220** is connected to decrementer **244** by lines **246** and **248**. The decrementer **244** receives a control input from a microprocessor system clock on line **250**.

The DMA CPU **72** controls itself and has the ability to fetch and execute instructions. It operates as a co-processor to the main CPU **70** (FIG. **2**) for time specific processing.

FIG. **6** shows how the microprocessor **50** is connected to an electrically programmable read only memory (EPROM) **260** by reconfiguring the data lines **52** so that some of the data lines **52** are input lines and some of them are output lines. Data lines **52** D0–D7 provide data to and from corresponding data terminals **262** of the EPROM **260**. Data lines **52** D9–D18 provide addresses to address terminals **264** of the EPROM **260**. Data lines **52** D19–D31 provide inputs from the microprocessor **50** to memory and I/O decode logic **266**. RAS 0/1 control line **268** provides a control signal for determining whether the memory and I/O decode logic provides a DRAM RAS output on line **270** or a column enable output for the EPROM **260** on line **272**. Column address strobe terminal **60** of the microprocessor **50** provides an output enable signal on line **274** to the corresponding terminal **276** of the EPROM **260**.

FIGS. **7** and **8** show the front and back of a one card data processing system **280** incorporating the microprocessor **50**, MSM514258-10 type DRAMs **150** totalling 2 megabytes, a Motorola 50 MegaHertz crystal oscillator clock **282**. I/O circuits **284** and a 27256 type EPROM **260**. The I/O circuits **284** include a 74HC04 type high speed hex inverter circuit **286**, an IDT39C828 type 10-bit inverting buffer circuit **288**, an IDT39C822 type 10-bit inverting register circuit **290** and two IDT39C823 type 9-bit non-inverting register circuits **292**. The card **280** is completed with a MAX12V type DC-DC converter circuit **294**, 34-pin dual AMP type headers **296**, a coaxial female power connector **298**, and a 3-pin AMP right angle header **300**. The card **280** is a low cost, imbeddable product that can be incorporated in larger systems or used as an internal development tool.

The microprocessor **50** is a very high performance (50 MHz) RISC influenced 32-bit CPU designed to work closely with dynamic RAM. Clock for clock, the microprocessor **50** approaches the theoretical performance limits possible with a single CPU configuration. Eventually, the microprocessor **50** and any other processor is limited by the bus bandwidth and the number of bus paths. The critical conduit is between the CPU and memory.

One solution to the bus bandwidth/bus path problem is to integrate a CPU directly onto the memory chips, giving every memory a direct bus the CPU. FIG. **9** shows another microprocessor **310** that is provided integrally with 1 mega-

bit of DRAM **311** in a single integrated circuit **312**. Until the present invention this solution has not been practical, because most high performance CPUs require from 500,000 to 1,000,000 transistors and enormous die sizes just by themselves. The microprocessor **310** is equivalent to the microprocessor **50** in FIGS. **1–8**. The microprocessors **50** and **310** are the most transistor efficient high performance CPUs in existence, requiring fewer than 50,000 transistors for dual processors **70** and **72** (FIG. **2**) or **314** and **316** (less memory). The very high speed of the microprocessors **50** and **310** is to a certain extent a function of the small number of active devices. In essence, the less silicon gets in the way the faster the electrons can get where they are going.

The microprocessor **310** is therefore the only CPU suitable for integration on the memory chip die **312**. Some simple modifications to the basic microprocessor **50** to take advantage of the proximity to the DRAM array **311** can also increase the microprocessor **50** clock speed by **50** percent, and probably more.

The microprocessor **310** core on board the DRAM die **312** provides most of the speed and functionality required for a large group of applications from automotive to peripheral control. However the integrated CPU **310**/DRAM **311** concept has the potential to redefine significantly the way multiprocessor solutions can solve a spectrum of very compute intensive problems. The CPU **310**/DRAM **311** combination eliminates the Von Neumann bottleneck by distributing it across numerous CPU/DRAM chips **312**. The microprocessor **310** is a particularly good core for multiprocessing, since it was designed with the SDI targeting array in mind and provisions were made for efficient interprocessor communications.

Traditional multiprocessor implementations have been very expensive in addition to being unable to exploit fully the available CPU horsepower. Multiprocessor systems have typically been built up from numerous board level or box level computers. The result is usually an immense amount of hardware with corresponding wiring, power consumption and communications problems. By the time the systems are interconnected, as much as 50 percent of the bus speed has been utilized just getting through the interfaces.

In addition, multiprocessor system software has been scarce. A multiprocessor system can easily be crippled by an inadequate load-sharing algorithm in the system software which allows one CPU to do a great deal of work and the others to be idle. Great strides have been made recently in systems software, and even UNIX V.4 may be enhanced to support multiprocessing. Several commercial products from such manufacturers as DUAL Systems and UNISOFT do a credible job on 68030 type microprocessor systems now.

The microprocessor **310** architecture eliminates most of the interface friction, since up to 64 CPU **310**/RAM **311** processors should be able to intercommunicate without buffers or latches. Each chip **312** has about 40 MIPS raw speed, because placing the DRAM **311** next to the CPU **310** allows the microprocessor **310** instruction cycle to be cut in half, compared to the microprocessor **50**. A 64 chip array of these chips **312** is more powerful than any other existing computer. Such an array fits on a 3×5 card, cost less than a TAX machine, and draw about the same power as a small television.

Dramatic changes in price/performance always reshape existing applications and almost always create new ones. The introduction of microprocessors in the mid 1970s created video games, personal computers, automotive computers, electronically controlled appliances, and low cost computer peripherals.

The integrated circuit **312** will find applications in all of the above areas, plus create some new ones. A common generic parallel processing algorithm handles convolution, Fast Fourier Transform (FFT)/pattern recognition. Interesting product possibilities using the integrated circuit **312** include high speed reading machines, real-time speech recognition, spoken language translation, real-time robot vision, a product to identify people by their faces, and an automotive or aviation collision avoidance system.

A real time processor for enhancing high density television (HDTV) images, or compressing the HDTV information into a smaller bandwidth, would be very feasible. The load sharing in HDTV could be very straightforward. Splitting up the task according to color and frame would require 6, 9 or 12 processors. Practical implementation might require 4 meg RAMs integrated with the microprocessor **310**.

The microprocessor **310** has the following specifications:

CONTROL LINES
4—POWER/GROUND
1—CLOCK
32—DATA I/O
4—SYSTEM CONTROL
  EXTERNAL MEMORY FETCH
  EXTERNAL MEMORY FETCH AUTOINCREMENT X
  EXTERNAL MEMORY FETCH AUTOINCREMENT Y
  EXTERNAL MEMORY WRITE
  EXTERNAL MEMORY WRITE AUTOINCREMENT X
  EXTERNAL MEMORY WRITE AUTOINCREMENT Y
  EXTERNAL PROM FETCH
  LOAD ALL X REGISTERS
  LOAD ALL Y REGISTERS
  LOAD ALL PC REGISTERS
  EXCHANGE X AND Y
  INSTRUCTION FETCH
  ADD TO PC
  ADD TO X
  WRITE MAPPING REGISTER
  READ MAPPING REGISTER
REGISTER CONFIGURATION
MICROPROCESSOR **310** CPU **316** CORE
COLUMN LATCH1 (1024 BITS) 32×32 MUX
STACK POINTER (16 BITS)
COLUMN LATCH2 (1024 BITS) 32×32 MUX
RSTACK POINTER (16 BITS)
PROGRAM COUNTER 32 BITS
X0 REGISTER 32 BITS (ACTIVATED ONLY FOR ON-CHIP ACCESSES)
Y0 REGISTER 32 BITS (ACTIVATED ONLY FOR ON-CHIP ACCESSES)
LOOP COUNTER 32 BITS
DMA CPU **314** CORE
DMA PROGRAM COUNTER 24 BITS
INSTRUCTION REGISTER 32 BITS
I/O & RAM ADDRESS REGISTER 32 BITS
TRANSFER SIZE COUNTER 12 BITS
INTERVAL COUNTER 12 BITS

To offer memory expansion for the basic chip **312**, an intelligent DRAM can be produced. This chip will be optimized for high speed operation with the integrated circuit **312** by having three on-chip address registers: Program Counter, X Register and Y register. As a result, to access the intelligent DRAM no address is required, and a total access cycle could be as short as 10 nsec. Each

expansion DRAM would maintain its own copy of the three registers and would be identified by a code specifying its memory address Incrementing and adding to the three registers will actually take place on the memory chips A maximum of 64 intelligent DRAM peripherals would allow a large system to be created without sacrificing speed by introducing multiplexers or buffers

There are certain differences between the microprocessor **310** and the microprocessor **50** that arise from providing the microprocessor **310** on the same die **312** with the DRAM **311** Integrating the DRAM **311** allows architectural changes in the microprocessor **310** logic to take advantage of existing on-chip DRAM **311** circuitry Row and column design is inherent in memory architecture The DRAMs **311** access random bits in a memory array by first selecting a row of 1024 bits, storing them into a column latch and then selecting one of the bits as the data to be read or written

The time required to access the data is split between the row access and the column access Selecting data already stored in a column latch is faster than selecting a random bit by at least a factor of six The microprocessor **310** takes advantage of this high speed by creating a number of column latches and using them as caches and shift registers Selecting a new row of information may be thought of as performing a 1024-bit read or write with the resulting immense bus bandwidth

1 The microprocessor **50** treats its 32-bit instruction register **108** (see FIGS 2 and 4) as a cache for four 8-bit instructions Since the DRAM **311** maintains a 1024-bit latch for the column bits, the microprocessor **310** treats the column latch as a cache for 128 8-bit instructions Therefore the next instruction will almost always be already present in the cache Long loops within the cache are also possible and more useful than the 4 instruction loops in the microprocessor **50**.

2 The microprocessor **50** uses two 16x32-bit deep register arrays **74** and **134** (FIG 2) for the parameter stack and the return stack The microprocessor **310** creates two other 1024-bit column latches to provide the equivalent of two 32x32-bit arrays which can be accessed twice as fast as a register array

3 The microprocessor **50** has a DMA capability which can be used for I/O to a video shift register The microprocessor **310** uses yet another 1024-bit column latch as a long video shift register to drive a CRT display directly For color displays, three on-chip shift registers could also be used. These shift registers can transfer pixels at a maximum of 100 MHz.

4 The microprocessor **50** accesses memory via an external 32-bit bus Most of the memory **311** for the microprocessor **310** is on the same die **312** External access to more memory is made using an 8-bit bus The result is a smaller die, smaller package and lower power consumption than the microprocessor **50**

5 The microprocessor **50** consumes about a third of its operating power charging and discharging the I/O pins and associated capacitances The DRAMs **150** (FIG 8) connected to the microprocessor **50** dissipate most of their power in the I/O drivers A microprocessor **310** system will consume about one-tenth the power of a microprocessor **50** system, since having the DRAM **311** next to the processor **310** eliminates most of the external capacitances to be charged and discharged

6 Multiprocessing means splitting a computing task between numerous processors in order to speed up the solution The popularity of multiprocessing is limited by the expense of current individual processors as well as the limited interprocessor communications ability The microprocessor **310** is an excellent multiprocessor candidate since the chip **312** is a monolithic computer complete with memory, rendering it low-cost and physically compact

The shift registers implemented with the microprocessor **310** to perform video output can also be configured as interprocessor communication links The INMOS transputer attempted a similar strategy but at much lower speed and without the performance benefits inherent in the microprocessor **310** column latch architecture Serial I O is a prerequisite for many multiprocessor topologies because of the many neighbor processors which communicate A cube has 6 neighbors Each neighbor communicates using these lines:

DATA IN
CLOCK IN
READY FOR DATA
DATA OUT
DATA READY?
CLOCK OUT

A special start up sequence is used to initialize the on-chip DRAM **311** in each of the processors

The microprocessor **310** column latch architecture allows neighbor processors to deliver information directly to internal registers or even instruction caches of other chips **312** This technique is not used with existing processors, because it only improves performance in a tightly coupled DRAM system

7 The microprocessor **50** architecture offers two types of looping structures: LOOP-IF-DONE and MICRO-LOOP The former takes an 8-bit to 24-bit operand to describe the entry point to the loop address. The latter performs a loop entirely within the 4 instruction queue and the loop entry point is implied as the first instruction in the queue Loops entirely within the queue run without external instruction fetches and execute up to three times as fast as the long loop construct The microprocessor **310** retains both constructs with a few differences The microprocessor **310** microloop functions in the same fashion as the microprocessor **50** operation, except the queue is 1024-bits or 128 8-bit instructions long The microprocessor **310** microloop can therefore contain jumps, branches, calls and immediate operations not possible in the 4 8-bit instruction microprocessor **50** queue

Microloops in the microprocessor **50** can only perform simple block move and compare functions The larger microprocessor **310** queue allows entire digital signal processing or floating point algorithms to loop at high speed in the queue

The microprocessor **50** offers four instructions to redirect execution:

CALL
BRANCH
BRANCH-IF-ZERO
LOOP-IF-NOT-DONE

These instructions take a variable length address operand 8, 16 or 24 bits long The microprocessor **50** next address logic treats the three operands similarly by adding or subtracting them to the current program counter For the microprocessor **310** the 16 and 24-bit operands function in the same manner as the 16 and 24-bit operands in the microprocessor **50** The 8-bit class operands are reserved to operate entirely within the instruction queue Next address decisions can therefore be made quickly, because only 10 bits of addresses are affected rather than 32 There is no carry or borrow generated past the 10 bits

8 The microprocessor **310** CPU **316** resides on an already crowded DRAM die **312** To keep chip size as small as

possible, the DMA processor **72** of the microprocessor **50** has been replaced with a more traditional DMA controller **314**. DMA is used with the microprocessor **310** to perform the following functions:

Video output to a CRT

Multiprocessor serial communications

8-bit parallel I/O

The DMA controller **314** can maintain both serial and parallel transfers simultaneously. The following DMA sources and destinations are supported by the microprocessor **310**:

DESCRIPTION	I/O	LINES
1 Video shift register	OUTPUT	1 to 3
2 Multiprocessor serial	BOTH	6 lines/channel
3 8-bit parallel	BOTH	8 data, 4 control

The three sources use separate 1024-bit buffers and separate I/O pins. Therefore, all three may be active simultaneously without interference.

The microprocessor **310** can be implemented with either a single multiprocessor serial buffer or separate receive and sending buffers for each channel, allowing simultaneous bidirectional communications with six neighbors simultaneously.

FIGS. **10** and **11** provide details of the PROM DMA used in the microprocessor **50**. The microprocessor **50** executes faster than all but the fastest PROMs. PROMS are used in a microprocessor **50** system to store program segments and perhaps entire programs. The microprocessor **50** provides a feature on power-up to allow programs to be loaded from low-cost, slow speed PROMs into high speed DRAM for execution. The logic which performs this function is part of the DMA memory controller **118**. The operation is similar to DMA, but not identical, since four 8-bit bytes must be assembled on the microprocessor **50** chip, then written to the DRAM **150**.

The microprocessor **50** directly interfaces to DRAM **150** over a triple multiplexed data and address bus **350** which carries RAS addresses, CAS addresses and data. The EPROM **260**, on the other hand, is read with non-multiplexed busses. The microprocessor **50** therefore has a special mode which unmultiplexes the data and address lines to read 8 bits of EPROM data. Four 8-bit bytes are read in this fashion. The multiplexed bus **350** is turned back on, and the data is written to the DRAM **150**.

When the microprocessor **50** detects a RESET condition, the processor stops the main CPU **70** and forces a mode **0** (PROM LOAD) instruction into the DMA CPU **72** instruction register. The DMA instruction directs the memory controller to read the EPROM **260** data at 8 times the normal access time for memory. Assuming a 50 MHz microprocessor **50**, this means an access time of 320 nsec. The instruction also indicates:

The selection address of the EPROM **260** to be loaded

The number of 32-bit words to transfer

The DRAM **150** address to transfer into

The sequence of activities to transfer one 32-bit word from EPROM **260** to DRAM **150** are:

1. RAS goes low at **352**, latching the EPROM **260** select information from the high order address bits. The EPROM **260** is selected.
2. Twelve address bits (consisting of what is normally DRAM CAS addresses plus two byte select bits are placed on the bus **350** going to the EPROM **260** address pins. These signals will remain on the lines until the data from the EPROM **260** has been read into the microprocessor **50**. For the first byte, the byte select bits will be binary 00.
3. CAS goes low at **354**, enabling the EPROM **260** data onto the lower 8 bits of the external address/data bus **350**. NOTE: It is important to recognize that during this part of the cycle, the lower 8 bits of the external data/address bus are functioning as inputs, but the rest of the bus is still acting as outputs.
4. The microprocessor **50** latches these eight least significant bits internally and shifts them 8 bits left to shift them to the next significant byte position.
5. Steps 2, 3 and 4 are repeated with byte address 01.
6. Steps 2, 3 and 4 are repeated with byte address 10.
7. Steps 2, 3 and 4 are repeated with byte address 11.
8. CAS goes high at **356**, taking the EPROM **260** off the data bus.
9. RAS goes high at **358**, indicating the end of the EPROM **260** access.
10. RAS goes low at **360**, latching the DRAM select information from the high order address bits. At the same time, the RAS address bits are latched into the DRAM **150**. The DRAM **150** is selected.
11. CAS goes low at 362, latching the DRAM **150** CAS addresses.
12. The microprocessor **50** places the previously latched EPROM **260** 32-bit data onto the external address/data bus **350**. W goes low at **364**, writing the 32 bits into the DRAM **150**.
13. W goes high at **366**, CAS goes high at **368**. The process continues with the next word.

FIG. **12** shows details of the microprocessor **50** memory controller **118**. In operation, bus requests stay present until they are serviced. CPU **70** requests are prioritized at **370** in the order of: 1, Parameter Stack; 2, Return Stack; 3, Data Fetch; 4, Instruction Fetch. The resulting CPU request signal and a DMA request signal are supplied as bus requests to bus control **372**, which provides a bus grant signal at **374**. Internal address bus **136** and a DMA counter **376** provide inputs to a multiplexer **378**. Either a row address or a column address are provided as an output to multiplexed address bus **380** as an output from the multiplexer **378**. The multiplexed address bus **380** and the internal data bus **90** provide address and data inputs, respectively, to multiplexer **382**. Shift register **384** supplies row address strobe (RAS) 1 and 2 control signals to multiplexer **386** and column address strobe (CAS) 1 and 2 control signals to multiplexer **388** on lines **390** and **392**. The shift register **384** also supplies output enable (OE) and write (W) signals on lines **394** and **396** and a control signal on line **398** to multiplexer **382**. The shift register **384** receives a RUN signal on line **400** to generate a memory cycle and supplies a MEMORY READY signal on line **402** when an access is complete.

STACK/REGISTER ARCHITECTURE

Most microprocessors use on-chip registers for temporary storage of variables. The on-chip registers access data faster than off-chip RAM. A few microprocessors use an on-chip push down stack for temporary storage.

A stack has the advantage of faster operation compared to on-chip registers by avoiding the necessity to select source and destination registers. (A math or logic operation always uses the top two stack items as source and the top of stack as destination.) The stack's disadvantage is that it makes some operations clumsy. Some compiler activities in particular require on-chip registers for efficiency.

As shown in FIG 13, the microprocessor **50** provides both on-chip registers **134** and a stack **74** and reaps the benefits of both

BENEFITS:

1. Stack math and logic is twice as fast as those available on an equivalent register only machine. Most programmers and optimizing compilers can take advantage of this feature

2. Sixteen registers are available for on-chip storage of local variables which can transfer to the stack for computation. The accessing of variables is three to four times as fast as available on a strictly stack machine

The combined stack **74**/register **134** architecture has not been used previously due to inadequate understanding by computer designers of optimizing compilers and the mix of transfer versus math/logic instructions.

ADAPTIVE MEMORY CONTROLLER

A microprocessor must be designed to work with small or large memory configurations. As more memory loads are added to the data, address and control lines the switching speed of the signals slows down. The microprocessor **50** multiplexes the address/data bus three ways so timing between the phases is critical. A traditional approach to the problem allocates a wide margin of time between bus phases so that systems will work with small or large numbers of memory chips connected. A speed compromise of as much as 50% is required.

As shown in FIG 14 the microprocessor **50** uses a feedback technique to allow the processor to adjust memory bus timing to be fast with small loads and slower with large ones. The OUTPUT ENABLE (OE) line **152** from the microprocessor **50** is connected to all memories **150** on the circuit board. The loading on the output enable line **152** to the microprocessor **50** is directly related to the number of memories **150** connected. By monitoring how rapidly OE **152** goes high after a read, the microprocessor **50** is able to determine when the data hold time has been satisfied and place the next address on the bus.

The level of the OE line **152** is monitored by CMOS input buffer **410** which generates an internal READY signal on line **412** to the microprocessor's memory controller. Curves **414** and **416** of the FIG 15 graph show the difference in rise time likely to be encountered from a lightly to heavily loaded memory system. When the OE line **152** has reached a predetermined level to generate the READY signal, driver **418** generates an OUTPUT ENABLE signal on OE line **152**.

SKIP WITHIN THE INSTRUCTION CACHE

The microprocessor **50** fetches four 8-bit instructions each memory cycle and stores them in a 32-bit instruction register **108**, as shown in FIG 16. A class of "test and skip" instructions can very rapidly execute a very fast jump operation within the four instruction cache.

SKIP CONDITIONS:

Always

ACC non-zero

ACC negative

Carry flag equal logic one

Never

ACC equal zero

ACC positive

Carry flag equal logic zero

The SKIP instruction can be located in any of the four byte positions **420** in the 32-bit instruction register **108**. If the test is successful, SKIP will jump over the remaining one, two or three 8-bit instructions in the instruction register **108** and cause the next four-instruction group to be loaded into the register **108**. As shown the SKIP operation is implemented by resetting the 2-bit microinstruction counter **180** to zero on line **422** and simultaneously latching the next instruction group into the register **108**. Any instructions following the SKIP in the instruction register are overwritten by the new instructions and not executed.

The advantage of SKIP is that optimizing compilers and smart programmers can often use it in place of the longer conditional JUMP instruction. SKIP also makes possible microloops which exit when the loop counts down or when the SKIP jumps to the next instruction group. The result in very fast code.

Other machines (such as the PDP-8 and Data General NOVA) provide the ability to skip a single instruction. The microprocessor **50** provides the ability to skip up to three instructions.

MICROLOOP IN THE INSTRUCTION CACHE

The microprocessor **50** provides the MICROLOOP instruction to execute repetitively from one to three instructions residing in the instruction register **108**. The microloop instruction works in conjunction with the LOOP COUNTER **92** (FIG 2) connected to the internal data bus **90**. To execute a microloop, the program stores a count in LOOP COUNTER **92**. MICROLOOP may be placed in the first, second, third, or last byte **420** of the instruction register **108**. If placed in the first position execution will just create a delay equal to the number stored in LOOP COUNTER **92** times the machine cycle. If placed in the second, third, or last byte **420**, when the microloop instruction is executed, it will test the LOOP COUNT for zero. If zero, execution will continue with the next instruction. If not zero, the LOOP COUNTER **92** is decremented and the 2-bit microinstruction counter is cleared causing the preceding instructions in the instruction register to be executed again.

Microloop is useful for block move and search operations. By executing a block move completely out of the instruction register **108** the speed of the move is doubled, since all memory cycles are used by the move rather than being shared with instruction fetching. Such a hardware implementation of microloops is much faster than conventional software implementation of a comparable function.

OPTIMAL CPU CLOCK SCHEME

The designer of a high speed microprocessor must produce a product which operate over wide temperature ranges, wide voltage swings, and wide variations in semiconductor processing. Temperature, voltage, and process all affect transistor propagation delays. Traditional CPU designs are done so that with the worse case of the three parameters, the circuit will function at the rated clock speed. The result are designs that must be clocked a factor of two slower than their maximum theoretical performance so they will operate properly in worse case conditions.

The microprocessor **50** uses the technique shown in FIGS **17–19** to generate the system clock and its required phases. Clock circuit **430** is the familiar "ring oscillator" used to test process performance. The clock is fabricated on the same silicon chip as the rest of the microprocessor **50**.

The ring oscillator frequency is determined by the parameters of temperature, voltage, and process. At room temperature, the frequency will be in the neighborhood of 100 MHZ. At 70 degrees Centigrade, the speed will be 50 MHZ. The ring oscillator **430** is useful as a system clock, with its stages **431** producing phase 0-phase 3 outputs **433** shown in FIG 19, because its performance tracks the parameters which similarly affect all other transistors on the same silicon die. By deriving system timing from the ring

oscillator **430**. CPU **70** will always execute at the maximum frequency possible but never too fast. For example, if the processing of a particular die is not good resulting in slow transistors, the latches and gates on the microprocessor **50** will operate slower than normal. Since the microprocessor **50** ring oscillator clock **430** is made from the same transistors on the same die as the latches and gates, it too will operate slower (oscillating at a lower frequency) providing compensation which allows the rest of the chip's logic to operate properly.

ASYNCHRONOUS/SYNCHRONOUS CPU

Most microprocessors derive all system timing from a single clock. The disadvantage is that different parts of the system can slow all operations. The microprocessor **50** provides a dual-clock scheme as shown in FIG. **17**, with the CPU **70** operating a synchronously to I/O interface **432** forming part of memory controller **118** (FIG. **2**) and the I/O interface **432** operating synchronously with the external world of memory and I/O devices. The CPU **70** executes at the fastest speed possible using the adaptive ring counter clock **430**. Speed may vary by a factor of four depending upon temperature, voltage, and process. The external world must be synchronized to the microprocessor **50** for operations such as video display updating and disc drive reading and writing. This synchronization is performed by the I/O interface **432**, speed of which is controlled by a conventional crystal clock **434**. The interface **432** processes requests for memory accesses from the microprocessor **50** and acknowledges the presence of I/O data. The microprocessor **50** fetches up to four instructions in a single memory cycle and can perform much useful work before requiring another memory access. By decoupling the variable speed of the CPU **70** from the fixed speed of the I/O interface **432** optimum performance can be achieved by each. Recoupling between the CPU **70** and the interface **432** is accomplished with handshake signals on lines **436**, with data/addresses passing on bus **90**, **136**.

ASYNCHRONOUS/SYNCHRONOUS CPU IMBEDDED ON A DRAM CHIP

System performance is enhanced even more when the DRAM **311** and CPU **314** (FIG. **9**) are located on the same die. The proximity of the transistors means that DRAM **311** and CPU **314** parameters will closely follow each other. At room temperature not only would the CPU **314** execute at 100 MHZ, but the DRAM **311** would access fast enough to keep up. The synchronization performed by the I/O interface **432** would be for DMA and reading and writing I/O ports. In some systems (such as calculators) no I/O synchronization at all would be required, and the I/O clock would be tied to the ring counter clock.

VARIABLE WIDTH OPERANDS

Many microprocessors provide variable width operands. The microprocessor **50** handles operands of 8, 16, or 24 bits using the same op-code. FIG. **20** shows the 32-bit instruction register **108** and the 2-bit microinstruction register **180** which selects the 8-bit instruction. Two classes of microprocessor **50** instructions can be greater than 8-bits, JUMP class and IMMEDIATE. A JUMP or IMMEDIATE op-code is 8-bits but the operand can be 8, 16 or 24 bits long. This magic is possible because operands must be right justified in the instruction register. This means that the least significant bit of the operand is always located in the least significant bit of the instruction register. The microinstruction counter **180** selects which 8-bit instruction to execute. If a JUMP or IMMEDIATE instruction is decoded, the state of the 2-bit microinstruction counter selects the required 8, 16, or 24 bit operand onto the address or data bus. The unselected 8-bit bytes are loaded with zeros by operation of decoder **440** and gates **442**. The advantage of this technique is the saving of a number of op-codes required to specify the different operand sizes in other microprocessors.

TRIPLE STACK CACHE

Computer performance is directly related to the system memory bandwidth. The faster the memories, the faster the computer. Fast memories are expensive, so techniques have been developed to move a small amount of high-speed memory around to the memory addresses where it is needed. A large amount of slow memory is constantly updated by the fast memory, giving the appearance of a large fast memory array. A common implementation of the technique is known as a high-speed memory cache. The cache may be thought of as fast acting shock absorber smoothing out the bumps in memory access. When more memory is required than the shock can absorb, it bottoms out and slow speed memory is accessed. Most memory operations can be handled by the shock absorber itself.

The microprocessor **50** architecture has the ALU **80** (FIG. **2**) directly coupled to the top two stack locations **76** and **78**. The access time of the stack **74** therefore directly affects the execution speed of the processor. The microprocessor **50** stack architecture is particularly suitable to a triple cache technique, shown in FIG. **21** which offers the appearance of a large stack memory operating at the speed of on-chip latches **450**. Latches **450** are the fastest form of memory device built on the chip, delivering data in as little as 3 nsec. However latches **450** require large numbers of transistors to construct. On-chip RAM **452** requires fewer transistors than latches but is slower by a factor of five (15 nsec access). Off-chip RAM **150** is the slowest storage of all. The microprocessor **50** organizes the stack memory hierarchy as three interconnected stacks **450**, **452** and **454**. The latch stack **450** is the fastest and most frequently used. The on-chip RAM stack **452** is next. The off-chip RAM stack **454** is slowest. The stack modulation determines the effective access time of the stack. If a group of stack operations never push or pull more than four consecutive items on the stack, operations will be entirely performed in the 3 nsec latch stack. When the four latches **456** are filled, the data in the bottom of the latch stack **450** is written to the top of the on-chip RAM stack **452**. When the sixteen locations **458** in the on-chip RAM stack **452** are filled, the data in the bottom of the on-chip RAM stack **452** is written to the top of the off-chip RAM stack **454**. When popping data off a full stack **450**, four pops will be performed before stack empty line **460** from the latch stack pointer **462** transfers data from the on-chip RAM stack **452**. By waiting for the latch stack **450** to empty before performing the slower on-chip RAM access, the high effective speed of the latches **456** are made available to the processor. The same approach is employed with the on-chip RAM stack **452** and the off-chip RAM stack **454**.

POLYNOMIAL GENERATION INSTRUCTION

Polynomials are useful for error correction, encryption, data compression and fractal generation. A polynomial is generated by a sequence of shift and exclusive OR operations. Special chips are provided for this purpose in the prior art.

The microprocessor **50** is able to generate polynomials at high speed without external hardware by slightly modifying how the ALU **80** works. As shown in FIG. **21**, a polynomial is generated by loading the "order" (also known as the feedback terms) into C Register **470**. The value thirty one (resulting in 32 iterations) is loaded into DOWN COUNTER **472**. A register **474** is loaded with zero. B register **476** is loaded with the starting polynomial value. When the POLY

instruction executes, C register **470** is exclusively ORed with A register **474** if the least significant bit of B register **476** is a one. Otherwise, the contents of the A register **474** passes through the ALU **80** unaltered. The combination of A and B is then shifted right (divided by 2) with shifters **478** and **480**. The operation automatically repeats the specified number of iterations, and the resulting polynomial is left in A register **474**.

FAST MULTIPLY

Most microprocessors offer a 16×16 or 32×32 bit multiply instruction. Multiply when performed sequentially takes one shift/add per bit, or 32 cycles for 32 bit data. The microprocessor **50** provides a high speed multiply which allows multiplication by small numbers using only a small number of cycles. FIG. **23** shows the logic used to implement the high speed algorithm. To perform a multiply, the size of the multiplier less one is placed in the DOWN COUNTER **472**. For a four bit multiplier, the number three would be stored in the DOWN COUNTER **472**. Zero is loaded into the A register **474**. The multiplier is written bit reversed into the B Register **476**. For example, a bit reversed five (binary 0101) would be written into B as 1010. The multiplicand is written into the C register **470**. Executing the FAST MULT instruction will leave the result in the A Register **474**, when the count has been completed. The fast multiply instruction is important because many applications scale one number by a much smaller number. The difference in speed between multiplying a 32×32 bit and a 32×4 bit is a factor of 8. If the least significant bit of the multiplier is a "ONE", the contents of the A register **474** and the C register **470** are added. If the least significant bit of the multiplier is a "ZERO", the contents of the A register are passed through the ALU **80** unaltered. The output of the ALU **80** is shifted left by shifter **482** in each iteration. The contents of the B register **476** are shifted right by the shifter **480** in each iteration.

INSTRUCTION EXECUTION PHILOSOPHY

The microprocessor **50** uses high speed D latches in most of the speed critical areas. Slower on-chip RAM is used as secondary storage.

The microprocessor **50** philosophy of instruction execution is to create a hierarchy of speed as follows:

Logic and D latch transfers	1 cycle	20 nsec
Math	2 cycles	40 nsec
Fetch/store on-chip RAM	2 cycles	40 nsec
Fetch/store in current RAS page	4 cycles	80 nsec
Fetch/store with RAS cycle	11 cycles	220 nsec

With a 50 MHZ clock, many operations can be performed in 20 nsec and almost everything else in 40 nsec.

To maximize speed, certain techniques in processor design have been used. They include:

Eliminating arithmetic operations on addresses

Fetching up to four instructions per memory cycle.

Pipelineless instruction decoding

Generating results before they are needed.

Use of three level stack caching

PIPELINE PHILOSOPHY

Computer instructions are usually broken down into sequential pieces, for example: fetch decode, register read, execute and store. Each piece will require a single machine cycle. In most Reduced Instruction Set Computer (RISC) chips, instruction require from three to six cycles.

RISC instructions are very parallel. For example, each of 70 different instructions in the SPARC (SUN Computer's RISC chip) has five cycles. Using a technique called "pipelining", the different phases of consecutive instructions can be overlapped.

To understand pipelining, think of building five residential homes. Each home will require in sequence, a foundation, framing, plumbing and wiring, roofing, and interior finish. Assume that each activity takes one week. To build one house will take five weeks.

But what if you want to build an entire subdivision? You have only one of each work crew, but when the foundation men finish on the first house, you immediately start them on the second one, and so on. At the end of five weeks, the first home is complete, but you also have five foundations. If you have kept the framing, plumbing, roofing, and interior guys all busy, from five weeks on, a new house will be completed each week.

This is the way a RISC chip like SPARC appears to execute an instruction in a single machine cycle. In reality, a RISC chip is executing one fifth of five instructions each machine cycle. And if five instructions stay in sequence, an instruction will be completed each machine cycle.

The problems with a pipeline are keeping the pipe full with instructions. Each time an out of sequence instruction such as a BRANCH or CALL occurs, the pipe must be refilled with the next sequence. The resulting dead time to refill the pipeline can become substantial when many IF THEN, ELSE statements or subroutines are encountered.

THE PIPELINE APPROACH

The microprocessor **50** has no pipeline as such. The approach of this microprocessor to speed is to overlap instruction fetching with execution of the previously fetched instruction(s). Beyond that, over half the instructions (the most common ones) execute entirely in a single machine cycle of 20 nsec. This is possible because:

1. Instruction decoding resolves in 2.5 nsec.
2. Incremented/decremented and some math values are calculated before they are needed, requiring only a latching signal to execute.
3. Slower memory is hidden from high speed operations by high-speed D latches which access in 4 nsec.

The disadvantage for this microprocessor is a more complex chip design process. The advantage for the chip user is faster ultimate throughput since pipeline stalls cannot exist. Pipeline synchronization with availability flag bits and other such pipeline handling is not required by this microprocessor.

For example, in some RISC machines an instruction which tests a status flag may have to wait for up to four cycles for the flag set by the previous instruction to be available to be tested. Hardware and software debugging is also somewhat easier because the user doesn't have to visualize five instructions simultaneously in the pipe.

OVERLAPPING INSTRUCTION FETCH/EXECUTE

The slowest procedure the microprocessor **50** performs is to access memory. Memory is accessed when data is read or written. Memory is also read when instructions are fetched. The microprocessor **50** is able to hide fetch of the next instruction behind the execution of the previously fetched instruction(s). The microprocessor **50** fetches instructions in 4-byte instruction groups. An instruction group may contain from one to four instructions. The amount of time required to execute the instruction group ranges from 4 cycles for simple instructions to 64 cycles for a multiply.

When a new instruction group is fetched, the microprocessor instruction decoder looks at the most significant bit of all four of the bytes. The most significant bit of an instruction determines if a memory access is required. For example, CALL, FETCH and STORE all require a memory access to

execute. If all four bytes have nonzero most significant bits, the microprocessor initiates the memory fetch of the next sequential 4-byte instruction group. When the last instruction in the group finishes executing, the next 4-byte instruction group is ready and waiting on the data bus needing only to be latched into the instruction register. If the 4-byte instruction group required four or more cycles to execute and the next sequential access was a column address strobe (CAS) cycle, the instruction fetch was completely overlapped with execution.

INTERNAL ARCHITECTURE

The microprocessor **50** architecture consists of the following:

PARAMETER STACK	<---> ALU <--->	Y REGISTER RETURN STACK
<---32 BITS---> 16 DEEP Used for math and logic		<---32 BITS---> 16 DEEP Used for subroutine and interrupt return addresses as well as local variables
Push down stack Can overflow into off-chip RAM		Push down stack Can overflow into off-chip RAM. Can also be accessed relative to top of stack
LOOP COUNTER	(32-bits, can decrement by 1) Used by class of test and loop instructions	
X REGISTER	(32-bits, can increment or decrement by 4). Used to point to RAM locations	
PROGRAM COUNTER	(32-bits, increments by 4). Points to 4-byte instruction groups in RAM	
INSTRUCTION REG	(32-Bits). Holds 4-byte instruction groups while they are being decoded and executed.	

MODE - A register with mode and status bits

MODE-BITS:

- Slow down memory accesses by 8 if "1". Run full speed if "0". (Provided for access to slow EPROM.)
- Divide the system clock by 1024 if "1" to reduce power consumption. Run full speed if "0". (On-chip counters slow down if this bit is set.)
- Enable external interrupt 1.
- Enable external interrupt 2
- Enable external interrupt 3
- Enable external interrupt 4
- Enable external interrupt 5
- Enable external interrupt 6
- Enable external interrupt 7

ON-CHIP MEMORY LOCATIONS:

MODE-BITS	
DMA-POINTER	
DMA-COUNTER	
STACK-POINTER	- Pointer into Parameter Stack.
STACK-DEPTH	- Depth of on-chip Parameter Stack
RSTACK-POINTER	- Pointer into Return Stack
RSTACK-DEPTH	- Depth of on-chip Return Stack

*Math and logic operations use the TOP item and NEXT to top Parameter Stack items as the operands. The result is pushed onto the Parameter Stack.
*Return addresses from subroutines are placed on the Return Stack. The Y REGISTER is used as a pointer to RAM locations. Since the Y REGISTER is the top item of the Return Stack, nesting of indices is straightforward.

ADDRESSING MODE HIGH POINTS

The data bus is 32-bits wide. All memory fetches and stores are 32-bits. Memory bus addresses are 30 bits. The least significant 2 bits are used to select one-of-four bytes in some addressing modes. The Program Counter, X Register, and Y Register are implemented as D latches with their outputs going to the memory address bus and the bus incrementer/decrementer. Incrementing one of these registers can happen quickly, because the incremented value has already rippled through the inc/dec logic and need only be clocked into the latch. Branches and Calls are made to 32-bit word boundaries.

INSTRUCTION SET

32-BIT INSTRUCTION FORMAT

The thirty two bit instructions are CALL, BRANCH, BRANCH-IF-ZERO, and LOOP-IF-NOT-DONE. These instructions require the calculation of an effective address. In many computers, the effective address is calculated by adding or subtracting an operand with the current Program Counter. This math operation requires from four to seven machine cycles to perform and can definitely bog down machine execution. The microprocessor's strategy is to perform the required math operation at assembly or linking time and do a much simpler "Increment to next page" or "Decrement to previous page" operation at run time. As a result, the microprocessor branches execute in a single cycle.

24-BIT OPERAND FORM:

Byte 1	Byte 2	Byte 3	Byte 4
WWWWWW XX	YYYYYYYY	YYYYYYYY	YYYYYYYY

With a 24-bit operand, the current page is considered to be defined by the most significant 6 bits of the Program Counter.

16-BIT OPERAND FORM: QQQQQQQQ-WWWWWW XX-YYYYYYYY-YYYYYYYY With a 16-bit operand, the current page is considered to be defined by the most significant 14 bits of the Program Counter.

8-BIT OPERAND FORM: QQQQQQQQ-QQQQQQQQ-WWWWWW XX-YYYYYYYY With an 8-bit operand, the current page is considered to be defined by the most significant 22 bits of the Program Counter.

QQQQQQQQ—Any 8-bit instruction

WWWWWW—Instruction op-code

XX—Select how the address bits will be used:

00—Make all high-order bits zero (Page zero addressing)

01—Increment the high-order bits (Use next page)

10—Decrement the high-order bits (Use previous page)

11—Leave the high-order bits unchanged (Use current page)

YYYYYYYY—The address operand field. This field is always shifted left two bits (to generate a word rather than byte address) and loaded into the Program Counter. The microprocessor instruction decoder figures out the width of the operand field by the location of the instruction op-code in the four bytes.

The compiler or assembler will normally use the shortest operand required to reach the desired address so that the leading bytes can be used to hold other instructions. The effective address is calculated by combining:

The current Program Counter,

The 8, 16, or 24 bit address operand in the instruction.

Using one of the four allowed addressing modes.

EXAMPLES OF EFFECTIVE ADDRESS CALCULATION

Example 1

Byte 1	Byte 2	Byte 3	Byte 4
QQQQQQQQ	QQQQQQQQ	00000011	10011000

The QQQQQQQQs in Byte 1 and 2 indicate space in the 4-byte memory fetch which could be hold two other

instructions to be executed prior to the CALL instruction. Byte 3 indicates a CALL instruction (six zeros) in the current page (indicated by the 11 bits) Byte 4 indicates that the hexadecimal number **98** will be forced into the Program Counter bits **2** through **10** (Remember, a CALL or BRANCH always goes to a word boundary so the two least significant bits are always set to zero). The effect of this instruction would be to CALL a subroutine at WORD location HEX 98 in the current page. The most significant 22 bits of the Program Counter define the current page and will be unchanged

Example 2

Byte 1	Byte 2	Byte 3	Byte 4
000001 01	00000001	00000000	00000000

If we assume that the Program Counter was HEX 0000 0156 which is binary:

00000000 00000000 00000001 01010110=OLD PROGRAM COUNTER

Byte 1 indicates a BRANCH instruction op code (000001) and "01" indicates select the next page Byte 2 3, and 4 are the address operand These 24-bits will be shifted to the left two places to define a WORD address HEX 0156 shifted left two places is HEX 0558 Since this is a 24-bit operand instruction the most significant 6 bits of the Program Counter define the current page These six bits will be incremented to select the next page Executing this instruction will cause the Program Counter to be loaded with HEX 0400 0558 which is binary:

00000100 00000000 00000101 01011000=NEW PROGRAM COUNTER

INSTRUCTIONS

CALL-LONG

0000 00XX-YYYYYYYY-YYYYYYYY-YYYYYYYY

Load the Program Counter with the effective WORD address specified Push the current PC contents onto the RETURN STACK.

OTHER EFFECTS: CARRY or modes, no effect May cause Return Stack to force an external memory cycle if on-chip Return Stack is full

BRANCH

0000 01XX-YYYYYYYY-YYYYYYYY-YYYYYYYY

Load the Program Counter with the effective WORD address specified

OTHER EFFECTS: NONE

BRANCH-IF-ZERO

0000 10XX-YYYYYYYY-YYYYYYYY-YYYYYYYY

Test the TOP value on the Parameter Stack If the value is equal to zero, load the Program Counter with the effective WORD address specified If the TOP value is not equal to zero increment the Program Counter and fetch and execute the next instruction.

OTHER EFFECTS: NONE

LOOP-IF-NOT-DONE

0000 11YY-(XXXX XXXX)-(XXXX XXXX)-(XXXX XXXX)

If the LOOP COUNTER is not zero, load the Program Counter with the effective WORD address specified. If the LOOP COUNTER is zero, decrement the LOOP COUNTER, increment the Program Counter and fetch and execute the next instruction.

OTHER EFFECTS: NONE

8-BIT INSTRUCTIONS PHILOSOPHY

Most of the work in the microprocessor **50** is done by the 8-bit instructions Eight bit instructions are possible with the microprocessor because of the extensive use of implied stack addressing Many 32-bit architectures use 8-bits to specify the operation to perform but use an additional 24-bits to specify two sources and a destination

For math and logic operations, the microprocessor **50** exploits the inherent advantage of a stack by designating the source operand(s) as the top stack item and the next stack item The math or logic operation is performed, the operands are popped from the stack, and the result is pushed back on the stack The result is a very efficient utilization of instruction bits as well as registers A comparable situation exists between Hewlett Packard calculators (which use a stack) and Texas Instrument calculators which don't The identical operation on an HP will require one half to one third the keystrokes of the TI.

The availability of 8-bit instructions also allows another architectural innovation, the fetching of four instructions in a single 32-bit memory cycle The advantages of fetching multiple instructions are:

Increased execution speed even with slow memories,

Similar performance to the Harvard (separate data and instruction busses) without the expense

Opportunities to optimize groups of instructions.

The capability to perform loops within this mini-cache The microloops inside the four instruction group are effective for searches and block moves

SKIP INSTRUCTIONS

The microprocessor **50** fetches instructions in 32-bit chunks called 4-byte instruction groups These four bytes may contain four 8-bit instructions or some mix of 8-bit and 16 or 24-bit instructions. SKIP instructions in the microprocessor skip any remaining instructions in a 4-byte instruction group and cause a memory fetch to get the next 4-byte instruction group Conditional SKIPs when combined with 3-byte BRANCHES will create conditional BRANCHES SKIPs may also be used in situations when no use can be made of the remaining bytes in a 4-instruction group A SKIP executes in a single cycle, whereas a group of three NOPs would take three cycles

SKIP-ALWAYS—Skip any remaining instructions in this 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group

SKIP-IF-ZERO—If the TOP item of the Parameter Stack is zero skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the TOP item is not zero execute the next sequential instruction

SKIP-IF-POSITIVE—If the TOP item of the Parameter Stack has a the most significant bit (the sign bit) equal to "0", skip any remaining instructions in the 4-byte instruction group Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group If the TOP item is not "0", execute the next sequential instruction

SKIP-IF-NO-CARRY—If the CARRY flag from a SHIFT or arithmetic operation is not equal to "1", skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group If the CARRY is equal to "1" execute the next sequential instruction

SKIP-NEVER (NOP) execute the next sequential instruction. (Delay one machine cycle)

SKIP-IF-NOT-ZERO—If the TOP item on the Parameter Stack is not equal to "0" skip any remaining instructions

in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the TOP item is equal "0", execute the next sequential instruction.

SKIP-IF-NEGATIVE—If the TOP item on the Parameter Stack has its most significant bit (sign bit) set to "1" skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the TOP item has its most significant bit set to "0", execute the next sequential instruction.

SKIP-IF-CARRY—If the CARRY flag is set to "1" as a result of SHIFT or arithmetic operation, skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the CARRY flag is "0" execute the next sequential instruction.

MICROLOOPS

Microloops are a unique feature of the microprocessor architecture which allows controlled looping within a 4-byte instruction group. A microloop instruction tests the LOOP COUNTER for "0" and may perform an additional test. If the LOOP COUNTER is not "0" and the test is met, instruction execution continues with the first instruction in the 4-byte instruction group, and the LOOP COUNTER is decremented. A microloop instruction will usually be the last byte in a 4-byte instruction group, but it can be any byte. If the LOOP COUNTER is "0" or the test is not met, instruction execution continues with the next instruction. If the microloop is the last byte in the 4-byte instruction group, the most significant 30-bits of the Program Counter are incremented and the next 4-byte instruction group is fetched from memory. On a termination of the loop on LOOP COUNTER equal to "0", the LOOP COUNTER will remain at "0". Microloops allow short iterative work such as moves and searches to be performed without slowing down to fetch instructions from memory.

EXAMPLE

Byte 1 FETCH-VIA-X-AUTO-INCREMENT	Byte 2 STORE-VIA-Y-AUTOINCREMENT
Byte 3 ULOOP-UNTIL-DONE	Byte 4 QQQQQQQQ

This example will perform a block move. To initiate the transfer, X will be loaded with the starting address of the source. Y will be loaded with the starting address of the destination. The LOOP COUNTER will be loaded with the number of 32-bit words to move. The microloop will FETCH and STORE and count down the LOOP COUNTER until it reaches zero. QQQQQQQQ indicates any instruction can follow.

MICROLOOP INSTRUCTIONS

ULOOP-UNTIL-DONE—If the LOOP COUNTER is not "0" continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0", continue execution with the next instruction.

ULOOP-IF-ZERO—If the LOOP COUNTER is not "0" and the TOP item on the Parameter Stack is "0", continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the TOP item is "1" continue execution with the next instruction.

ULOOP-IF-POSITIVE—If the LOOP COUNTER is not "0" and the most significant bit (sign bit) is "0" continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the TOP item is "1" continue execution with the next instruction.

ULOOP-IF-NOT-CARRY-CLEAR—If the LOOP COUNTER is not "0" and the floating point exponents found in TOP and NEXT are not aligned continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the exponents are aligned, continue execution with the next instruction. This instruction is specifically designed for combination with special SHIFT instructions to align two floating point numbers.

ULOOP-NEVER—(DECREMENT-LOOP-COUNTER) Decrement the LOOP COUNTER. Continue execution with the next instruction.

ULOOP-IF-NOT-ZERO—If the LOOP COUNTER is not "0" and the TOP item of the Parameter Stack is "0", continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the TOP item is "1" continue execution with the next instruction.

ULOOP-IF-NEGATIVE—If the LOOP COUNTER is not "0" and the most significant bit (sign bit) of the TOP item of the Parameter Stack is "1", continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the most significant bit of the Parameter Stack is "0", continue execution with the next instruction.

ULOOP-IF-CARRY-SET—If the LOOP COUNTER is not "0" and the exponents of the floating point numbers found in TOP and NEXT are not aligned, continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the exponents are aligned, continue execution with the next instruction.

RETURN FROM SUBROUTINE OR INTERRUPT

Subroutine calls and interrupt acknowledgements cause a redirection of normal program execution. In both cases the current Program Counter is pushed onto the Return Stack so the microprocessor can return to its place in the program after executing the subroutine or interrupt service routine.

NOTE: When a CALL to subroutine or interrupt is acknowledged the Program Counter has already been incremented and is pointing to the 4-byte instruction group following the 4-byte group currently being executed. The instruction decoding logic allows the microprocessor to perform a test and execute a return conditional on the outcome of the test in a single cycle. A RETURN pops an address from the Return Stack and stores it to the Program Counter.

RETURN INSTRUCTIONS

RETURN-ALWAYS—Pop the top item from the Return Stack and transfer it to the Program Counter.

RETURN-IF-ZERO—If the TOP item on the Parameter Stack is "0" pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

RETURN-IF-POSITIVE—If the most significant bit (sign bit) of the TOP item on the Parameter Stack is a "0", pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

RETURN-IF-CARRY-CLEAR—If the exponents of the floating point numbers found in TOP and NEXT are not aligned, pop the top item from the Return Stack and

transfer it to the Program Counter. Otherwise execute the next instruction.

RETURN-NEVER (NOP)—Execute the next instruction.

RETURN-IF-NOT-ZERO—If the TOP item on the Parameter Stack is not "0", pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

RETURN-IF-NEGATIVE—If the most significant bit (sign bit) of the TOP item on the Parameter Stack is a "1", pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

RETURN-IF-CARRY-SET—If the exponents of the floating point numbers found in TOP and NEXT are aligned, pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

HANDLING MEMORY FROM DYNAMIC RAM

The microprocessor **50**, like any RISC type architecture, is optimized to handle as many operations as possible on-chip for maximum speed. External memory operations take from 80 nsec. to 220 nsec. compared with on-chip memory speeds of from 4 nsec. to 30 nsec. There are times when external memory must be accessed.

External memory is accessed using three registers:

X-REGISTER—A 30-bit memory pointer which can be used for memory access and simultaneously incremented or decremented.

Y-REGISTER—A 30-bit memory pointer which can be used for memory access and simultaneously incremented or decremented.

PROGRAM-COUNTER—A 30-bit memory pointer normally used to point to 4-byte instruction groups. External memory may be accessed at addresses relative to the PC. The operands are sometimes called "Immediate" or "Literal" in other computers. When used as memory pointer, the PC is also incremented after each operation.

MEMORY LOAD & STORE INSTRUCTIONS

FETCH-VIA-X—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. X is unchanged.

FETCH-VIA-Y—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. Y is unchanged.

FETCH-VIA-X-AUTOINCREMENT—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. After fetching, increment the most significant 30 bits of X to point to the next 32-bit word address.

FETCH-VIA-Y-AUTOINCREMENT—Fetch the 32-bit memory content pointed to by Y and push it onto the Parameter Stack. After fetching, increment the most significant 30 bits of Y to point to the next 32-bit word address.

FETCH-VIA-X-AUTODECREMENT—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. After fetching, decrement the most significant 30 bits of X to point to the previous 32-bit word address.

FETCH-VIA-Y-AUTODECREMENT—Fetch the 32-bit memory content pointed to by Y and push it onto the Parameter Stack. After fetching, decrement the most significant 30 bits of Y to point to the previous 32-bit word address.

STORE-VIA-X—Pop the top item of the Parameter Stack and store it in the memory location pointed to by X. X is unchanged.

STORE-VIA-Y—Pop the top item of the Parameter Stack and store it in the memory location pointed to by Y. Y is unchanged.

STORE-VIA-X-AUTOINCREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by X. After storing, increment the most significant 30 bits of X to point to the next 32-bit word address.

STORE-VIA-Y-AUTOINCREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by Y. After storing, increment the most significant 30 bits of Y to point to the next 32-bit word address.

STORE-VIA-X-AUTODECREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by X. After storing, decrement the most significant 30 bits of X to point to the previous 32-bit word address.

STORE-VIA-Y-AUTODECREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by Y. After storing, decrement the most significant 30 bits of Y to point to the previous 32-bit word address.

FETCH-VIA-PC—Fetch the 32-bit memory content pointed to by the Program Counter and push it onto the Parameter Stack. After fetching, increment the most significant 30 bits of the Program Counter to point to the next 32-bit word address.

*NOTE: When this instruction executes, the PC is pointing to the memory location following the instruction. The effect is of loading a 32-bit immediate operand. This is an 8-bit instruction and therefore will be combined with other 8-bit instructions in a 4-byte instruction fetch. It is possible to have from one to four FETCH-VIA-PC instructions in a 4-byte instruction fetch. The PC increments after each execution of FETCH-VIA-PC, so it is possible to push four immediate operands on the stack. The four operands would be the found in the four memory locations following the instruction.

BYTE-FETCH-VIA-X—Fetch the 32-bit memory content pointed to by the most significant 30 bits of X. Using the two least significant bits of X, select one of four bytes from the 32-bit memory fetch, right justify the byte in a 32-bit field and push the selected byte preceded by leading zeros onto the Parameter Stack.

BYTE-STORE-VIA-X—Fetch the 32-bit memory content pointed to by the most significant 30 bits of X. Pop the TOP item from the Parameter Stack. Using the two least significant bits of X place the least significant byte into the 32-bit memory data and write the 32-bit entity back to the location pointed to by the most significant 30 bits of X.

OTHER EFFECTS OF MEMORY ACCESS INSTRUCTIONS:

Any FETCH instruction will push a value on the Parameter Stack **74**. If the on-chip stack is full, the stack will overflow into off-chip memory stack resulting in an additional memory cycle. Any STORE instruction will pop a value from the Parameter Stack **74**. If the on-chip stack is empty, a memory cycle will be generated to fetch a value from off-chip memory stack.

HANDLING ON-CHIP VARIABLES

High-level languages often allow the creation of LOCAL VARIABLES. These variables are used by a particular procedure and discarded. In cases of nested procedures, layers of these variables must be maintained. On-chip storage is up to five times faster than off-chip RAM, so a means of keeping local variables on-chip can make operations run faster. The microprocessor **50** provides the capability for both on-chip storage of local variables and nesting of multiple levels of variables through the Return Stack.

The Return Stack **134** is implemented as 16 on-chip RAM locations. The most common use for the Return Stack **134** is storage of return addresses from subroutines and interrupt calls. The microprocessor allows these 16 locations to also be used as addressable registers. The 16 locations may be read and written by two instructions which indicate a Return Stack relative address from 0–15. When high-level procedures are nested, the current procedure variables push the previous procedure variables further down the Return Stack **134**. Eventually, the Return Stack will automatically overflow into off-chip RAM.

ON-CHIP VARIABLE INSTRUCTIONS

READ-LOCAL-VARIABLE XXXX—Read the XXXXth location relative to the top of the Return Stack. (XXXX is a binary number from 0000–1111). Push the item read onto the Parameter Stack.

OTHER EFFECTS: If the Parameter Stack is full, the push operation will cause a memory cycle to be generated as one item of the stack is automatically stored to external RAM. The logic which selects the location performs a modulo **16** subtraction. If four local variables have been pushed onto the Return Stack, and an instruction attempts to READ the fifth item, unknown data will be returned.

WRITE-LOCAL-VARIABLE XXXX—Pop the TOP item of the Parameter Stack and write it into the XXXXth location relative to the top of the Return Stack. (XXXX is a binary number from 0000–1111.)

OTHER EFFECTS: If the Parameter Stack is empty, the pop operation will cause a memory cycle to be generated to fetch the Parameter Stack item from external RAM. The logic which selects the location performs a modulo **16** subtraction. If four local variables have been pushed onto the Return Stack, and an instruction attempts to WRITE to the fifth item, it is possible to clobber return addresses or wreak other havoc.

REGISTER AND FLIP-FLOP TRANSFER AND PUSH INSTRUCTIONS

DROP—Pop the TOP item from the Parameter Stack and discard it.

SWAP—Exchange the data in the TOP Parameter Stack location with the data in the NEXT Parameter Stack location.

DUP—Duplicate the TOP item on the Parameter Stack and push it onto the Parameter Stack.

PUSH-LOOP-COUNTER—Push the value in LOOP COUNTER onto the Parameter Stack.

POP-RSTACK-PUSH-TO-STACK—Pop the top item from the Return Stack and push it onto the Parameter Stack.

PUSH-X-REG—Push the value in the X Register onto the Parameter Stack.

PUSH-STACK-POINTER—Push the value of the Parameter Stack pointer onto the Parameter Stack.

PUSH-RSTACK-POINTER—Push the value of the Return Stack pointer onto the Return Stack.

PUSH-MODE-BITS—Push the value of the MODE REGISTER onto the Parameter Stack.

PUSH-INPUT—Read the 10 dedicated input bits and push the value (right justified and padded with leading zeros) onto the Parameter Stack.

SET-LOOP-COUNTER—Pop the TOP value from the Parameter Stack and store it into LOOP COUNTER.

POP-STACK-PUSH-TO-RSTACK—Pop the TOP item from the Parameter Stack and push it onto the Return Stack.

SET-X-REG—Pop the TOP item from the Parameter Stack and store it into the X Register.

SET-STACK-POINTER—Pop the TOP item from the Parameter Stack and store it into the Stack Pointer.

SET-RSTACK-POINTER—Pop the TOP item from the Parameter Stack and store it into the Return Stack Pointer.

SET-MODE-BITS—Pop the TOP value from the Parameter Stack and store it into the MODE BITS.

SET-OUTPUT—Pop the TOP item from the Parameter Stack and output it to the 10 dedicated output bits.

OTHER EFFECTS: Instructions which push or pop the Parameter Stack or Return Stack may cause a memory cycle as the stacks overflow back and forth between on-chip and off-chip memory.

LOADING A SHORT LITERAL

A special case of register transfer instruction is used to push an 8-bit literal onto the Parameter Stack. This instruction requires that the 8-bits to be pushed reside in the last byte of a 4-byte instruction group. The instruction op-code loading the literal may reside in ANY of the other three bytes in the instruction group.

EXAMPLE

BYTE 1 LOAD-SHORT-LITERAL BYTE 4 00001111	BYTE 2 QQQQQQQQ	BYTE 3 QQQQQQQQ

In this example, QQQQQQQQ indicates any other 8-bit instruction. When Byte 1 is executed, binary 00001111(HEX 0f) from Byte 4 will be pushed (right justified and padded by leading zeros) onto the Parameter Stack. Then the instructions in Byte 2 and Byte 3 will execute. The microprocessor instruction decoder knows not to execute Byte 4. It is possible to push three identical 8-bit values as follows.

BYTE 1 LOAD-SHORT-LITERAL BYTE 3 LOAD-SHORT-LITERAL SHORT-LITERAL-INSTRUCTION	BYTE 2 LOAD-SHORT-LITERAL BYTE 4 00001111

LOAD-SHORT-LITERAL—Push the 8-bit value found in Byte 4 of the current 4-byte instruction group onto the Parameter Stack.

LOGIC INSTRUCTIONS

Logical and math operations used the stack for the source of one or two operands and as the destination for results. The stack organization is a particularly convenient arrangement for evaluating expressions. TOP indicates the top value on the Parameter Stack **74**. NEXT indicates the next to top value on the Parameter Stack **74**.

AND—Pop TOP and NEXT from the Parameter Stack, perform the logical AND operation on these two operands, and push the result onto the Parameter Stack.

OR—Pop TOP and NEXT from the Parameter Stack, perform the logical OR operation on these two operands, and push the result onto the Parameter Stack.

XOR—Pop TOP and NEXT from the Parameter Stack, perform the logical exclusive OR on these two operands, and push the result onto the Parameter Stack.

BIT-CLEAR—Pop TOP and NEXT from the Parameter Stack, toggle all bits in NEXT, perform the logical AND operation on TOP and push the result onto the Parameter Stack. (Another way of understanding this instruction is thinking of it as clearing all bits in TOP that are set in NEXT.)

MATH INSTRUCTIONS

Math instruction pop the TOP item and NEXT to top item of the Parameter Stack 74 to use as the operands. The results are pushed back on the Parameter Stack. The CARRY flag is used to latch the "33rd bit" of the ALU result

ADD—Pop the TOP item and NEXT to top item from the Parameter Stack, add the values together and push the result back on the Parameter Stack. The CARRY flag may be changed.

ADD-WITH-CARRY—Pop the TOP item and the NEXT to top item from the Parameter Stack, add the values together. If the CARRY flag is "1" increment the result. Push the ultimate result back on the Parameter Stack. The CARRY flag may be changed

ADD-X—Pop the TOP item from the Parameter Stack and read the third item from the top of the Parameter Stack. Add the values together and push the result back on the Parameter Stack. The CARRY flag may be changed

SUB—Pop the TOP item and NEXT to top item from the Parameter Stack, Subtract NEXT from TOP and push the result back on the Parameter Stack. The CARRY flag may be changed.

SUB-WITH-CARRY—Pop the TOP item and NEXT to top item from the Parameter Stack. Subtract NEXT from TOP. If the CARRY flag is "1" increment the result. Push the ultimate result back on the Parameter Stack. The CARRY flag may be changed

SUB-X—"

SIGNED-MULT-STEP—

UNSIGNED-MULT-STEP—

SIGNED-FAST-MULT—

FAST-MULT-STEP—

UNSIGNED-DIV-STEP—

GENERATE-POLYNOMIAL—

ROUND—

COMPARE—Pop the TOP item and NEXT to top item from the Parameter Stack. Subtract NEXT from TOP. If the result has the most significant bit equal to "0" (the result is positive), push the result onto the Parameter Stack. If the result has the most significant bit equal to "1" (the result is negative), push the old value of TOP onto the Parameter Stack. The CARRY flag may be affected

SHIFT/ROTATE

SHIFT-LEFT—Shift the TOP Parameter Stack item left one bit. The CARRY flag is shifted into the least significant bit of TOP

SHIFT-RIGHT—Shift the TOP Parameter Stack item right one bit. The least significant bit of TOP is shifted into the CARRY flag. Zero is shifted into the most significant bit of TOP

DOUBLE-SHIFT-LEFT—Treating the TOP item of the Parameter Stack as the most significant word of a 64-bit number and the NEXT stack item as the least significant word, shift the combined 64-bit entity left one bit. The CARRY flag is shifted into the least significant bit of NEXT

DOUBLE-SHIFT-RIGHT—Treating the TOP item of the Parameter Stack as the most significant word of a 64-bit number and the NEXT stack item as the least significant word, shift the combined 64-bit entity right one bit. The least significant bit of NEXT is shifted into the CARRY flag. Zero is shifted into the most significant bit of TOP

OTHER INSTRUCTIONS

FLUSH-STACK—Empty all on-chip Parameter Stack locations into off-chip RAM. (This instruction is useful for multitasking applications). This instruction accesses a counter which holds the depth of the on-chip stack and can require from none to 16 external memory cycles

FLUSH-RSTACK—Empty all on-chip Return Stack locations into off-chip RAM. (This instruction is useful for multitasking applications). This instruction accesses a counter which holds the depth of the on-chip Return Stack and can require from none to 16 external memory cycles

It should further be apparent to those skilled in the art that various changes in form and details of the invention as shown and described may be made. It is intended that such changes be included within the spirit and scope of the claims appended hereto.

What is claimed is:

1. A microprocessor system, comprising a single integrated circuit including a central processing unit and an entire ring oscillator variable speed system clock in said single integrated circuit and connected to said central processing unit for clocking said central processing unit, said central processing unit and said ring oscillator variable speed system clock each including a plurality of electronic devices correspondingly constructed of the same process technology with corresponding manufacturing variations, a processing frequency capability of said central processing unit and a speed of said ring oscillator variable speed system clock varying together due to said manufacturing variations and due to at least operating voltage and temperature of said single integrated circuit; an on-chip input/output interface connected to exchange coupling control signals, addresses and data with said central processing unit; and a second clock independent of said ring oscillator variable speed system clock connected to said input/output interface.

2. The microprocessor system of claim 1 in which said second clock is a fixed frequency clock.

3. In a microprocessor integrated circuit, a method for clocking the microprocessor within the integrated circuit, comprising the steps of:

providing an entire ring oscillator system clock constructed of electronic devices within the integrated circuit, said electronic devices having operating characteristics which will because said entire ring oscillator system clock and said microprocessor are located within the same integrated circuit, vary together with operating characteristics of electronic devices included within the microprocessor;

using the ring oscillator system clock for clocking the microprocessor said microprocessor operating at a variable processing frequency dependent upon a variable speed of said ring oscillator system clock;

providing an on chip input/output interface for the microprocessor integrated circuit; and

clocking the input/output interface with a second clock independent of the ring oscillator system clock.

4. The method of claim 3 in which the second clock is a fixed frequency clock.

5. The method of claim 3 further including the step of:

transferring information to and from said microprocessor in synchrony with said ring oscillator system clock.

6. A microprocessor system comprising:

a central processing unit disposed upon an integrated circuit substrate, said central processing unit operating at a processing frequency and being constructed of a first plurality of electronic devices;

an entire oscillator disposed upon said integrated circuit substrate and connected to said central processing unit, said oscillator clocking said central processing unit at a clock rate and being constructed of a second plurality of electronic devices thus varying the processing frequency of said first plurality of electronic devices and

the clock rate of said second plurality of electronic devices in the same way as a function of parameter variation in one or more fabrication or operational parameters associated with said integrated circuit substrate thereby enabling said processing frequency to track said clock rate in response to said parameter variation;

an on-chip input output interface connected between said said central processing unit and an external memory bus, for facilitating exchanging coupling control signals, addresses and data with said central processing unit; and

an external clock independent of said oscillator, connected to said input/output interface wherein said external clock is operative at a frequency independent of a clock frequency of said oscillator

7 The microprocessor system of claim **6** wherein said one or more operational parameters include operating temperature of said substrate or operating voltage of said substrate.

8 The microprocessor system of claim **6** wherein said external clock comprises a fixed-frequency clock which operates synchronously relative to said oscillator

9 The microprocessor system of claim **6** wherein said oscillator comprises a ring oscillator

10 In a microprocessor system including a central processing unit, a method for clocking said central processing unit comprising the steps of:

providing said central processing unit upon an integrated circuit substrate said central processing unit being constructed of a first plurality of transistors and being operative at a processing frequency;

providing an entire variable speed clock disposed upon said integrated circuit substrate, said variable speed clock being constructed of a second plurality of transistors;

clocking said central processing unit at a clock rate using said variable speed clock with said central processing unit being clocked by said variable speed clock at a variable frequency dependent upon variation in one or more fabrication or operational parameters associated with said integrated circuit substrate said processing frequency and said clock rate varying in the same way relative to said variation in said one or more fabrication or operational parameters associated with said integrated circuit substrate;

connecting an on chip input/output interface between said central processing unit and an external memory bus, and exchanging coupling control signals addresses and data between said input/output interface and said central processing unit; and

clocking said input/output interface using an external clock wherein said external clock is operative at a frequency independent of a clock frequency of said oscillator

* * * * *

US005440749A

United States Patent [19]

Moore et al.

[11] Patent Number: 5,440,749

[45] Date of Patent: Aug. 8, 1995

[54] **HIGH PERFORMANCE, LOW COST MICROPROCESSOR ARCHITECTURE**

[75] Inventors: **Charles H. Moore**, Woodside; **Russell H. Fish, III**, Mt. View, both of Calif.

[73] Assignee: **Nanotronics Corporation**, Eagle Point, Oreg.

[21] Appl. No.: **389,334**

[22] Filed: **Aug. 3, 1989**

[51] **Int. Cl.**[6] **G06F 9/22**

[52] **U.S. Cl.** **395/800**; 364/931; 364/925.6; 364/937.1; 364/965.4; 364/232.8; 364/244.3

[58] **Field of Search** 395/425, 725, 775, 800

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,603,934	9/1971	Heath	364/DIG. 1
4,003,033	1/1977	O'Keefe et al.	364/200
4,037,090	7/1977	Raymond	364/200
4,042,972	8/1977	Grunes et al.	364/200
4,050,058	9/1977	Garlic	395/800
4,067,059	1/1978	Derchak	364/DIG. 1
4,079,455	3/1978	Ozga	395/800
4,110,822	8/1978	Porter	364/200
4,125,871	11/1978	Martin	364/DIG. 2
4,128,873	12/1978	Lamiaux	364/200
4,255,785	3/1981	Chamberlin	395/375
4,354,228	10/1982	Moore et al.	364/200
4,376,977	3/1983	Brunshorst	364/DIG. 1
4,382,279	5/1983	Mgon	364/200
4,403,303	9/1983	Howes et al.	364/900
4,450,519	5/1984	Guttag et al.	364/200
4,463,421	7/1984	Laws	395/325
4,538,239	8/1985	Magar	364/759
4,541,045	9/1985	Kromer	395/375
4,562,537	12/1985	Barnett et al.	395/375
4,577,282	3/1986	Caudel et al.	395/800
4,607,332	8/1986	Goldberg	364/900
4,626,988	12/1986	George et al.	364/200
4,649,471	3/1987	Briggs	395/325
4,665,495	5/1987	Thaden	345/185
4,709,329	11/1987	Hecker	395/275
4,713,749	12/1987	Magar et al.	395/375
4,714,994	12/1987	Oklobdzija et al.	395/375
4,720,812	1/1988	Kao et al.	395/700
4,772,888	9/1988	Kimura	340/825.5
4,777,591	10/1988	Chang et al.	395/800
4,787,032	11/1988	Culley et al.	364/200
4,803,621	2/1989	Kelly	395/400
4,860,198	8/1989	Takenaka	364/DIG. 1
4,870,562	9/1989	Kimoto	364/DIG. 1
4,931,986	6/1990	Daniel et al.	395/550
5,036,460	7/1991	Takahira	395/425
5,070,451	12/1991	Moore et al.	395/375
5,127,091	6/1992	Bonfarah	395/375

OTHER PUBLICATIONS

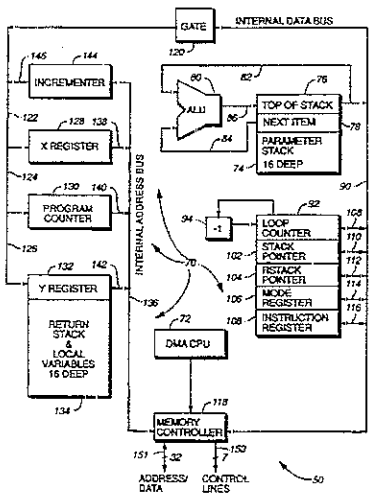
Intel 80386 Programmer's Reference Manual, 1986.

Primary Examiner—David Y. Eng
Attorney, Agent, or Firm—Cooley Godward Castro Huddleson & Tatum

[57] **ABSTRACT**

A microprocessor (**50**) includes a main central processing unit (CPU) (**70**) and a separate direct memory access (DMA) CPU (**72**) in a single integrated circuit making up the microprocessor (**50**). The main CPU (**70**) has a first 16 deep push down tack (**74**), which has a to item register (**76**) and a next item register (**78**), respectively connected to provide inputs to an arithmetic logic unit (ALU) (**80**) by lines (**82**) and (**84**). An output of the ALU (**80**) is connected to the top item register at (**82**) is also connected by line (**88**) to an internal data bus (**90**). CPU (**70**) is pipeline free. The simplified CPU (**70**) requires fewer transistors to implement than pipelined architectures, yet produces performance which matches or exceeds existing techniques. The DMA CPU (**72**) provides inputs to the memory controller (**118**) on line (**148**). The memory controller (**118**) is connected to a RAM by address/data bus (**150**) and control lines (**152**). The DMA CPU (**72**) enables the CPU (**70**) to execute instructions four times faster than the RAM speed by fetching four instructions in a single memory cycle

29 Claims, 19 Drawing Sheets



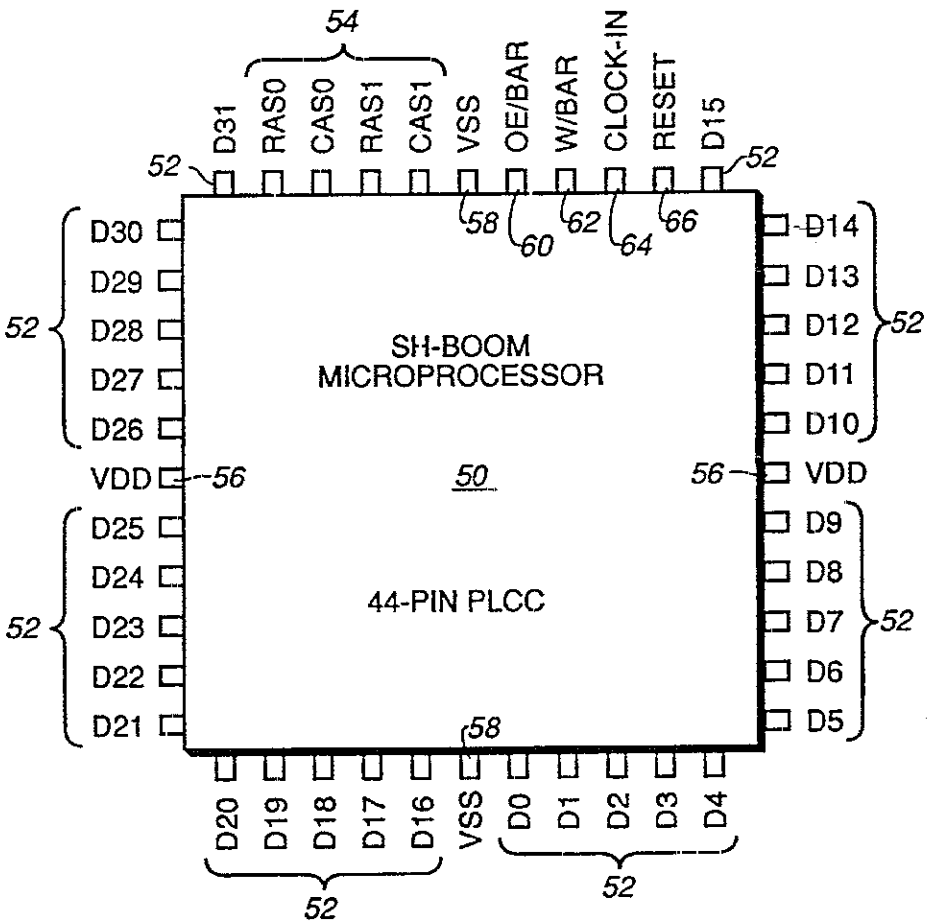

54
D31
RAS0
CAS0
RAS1
CAS1
VSS
OE/BAR
W/BAR
CLOCK-IN
RESET
D15
52
58
60
62
64
66
D30
D29
D28
D27
D26
VDD
56
SH-BOOM
MICROPROCESSOR
50
44-PIN PLCC
D14
D13
D12
D11
D10
D25
D24
D23
D22
D21
D9
D8
D7
D6
D5
D20
D19
D18
D17
D16
D0
D1
D2
D3
D4

FIG._1

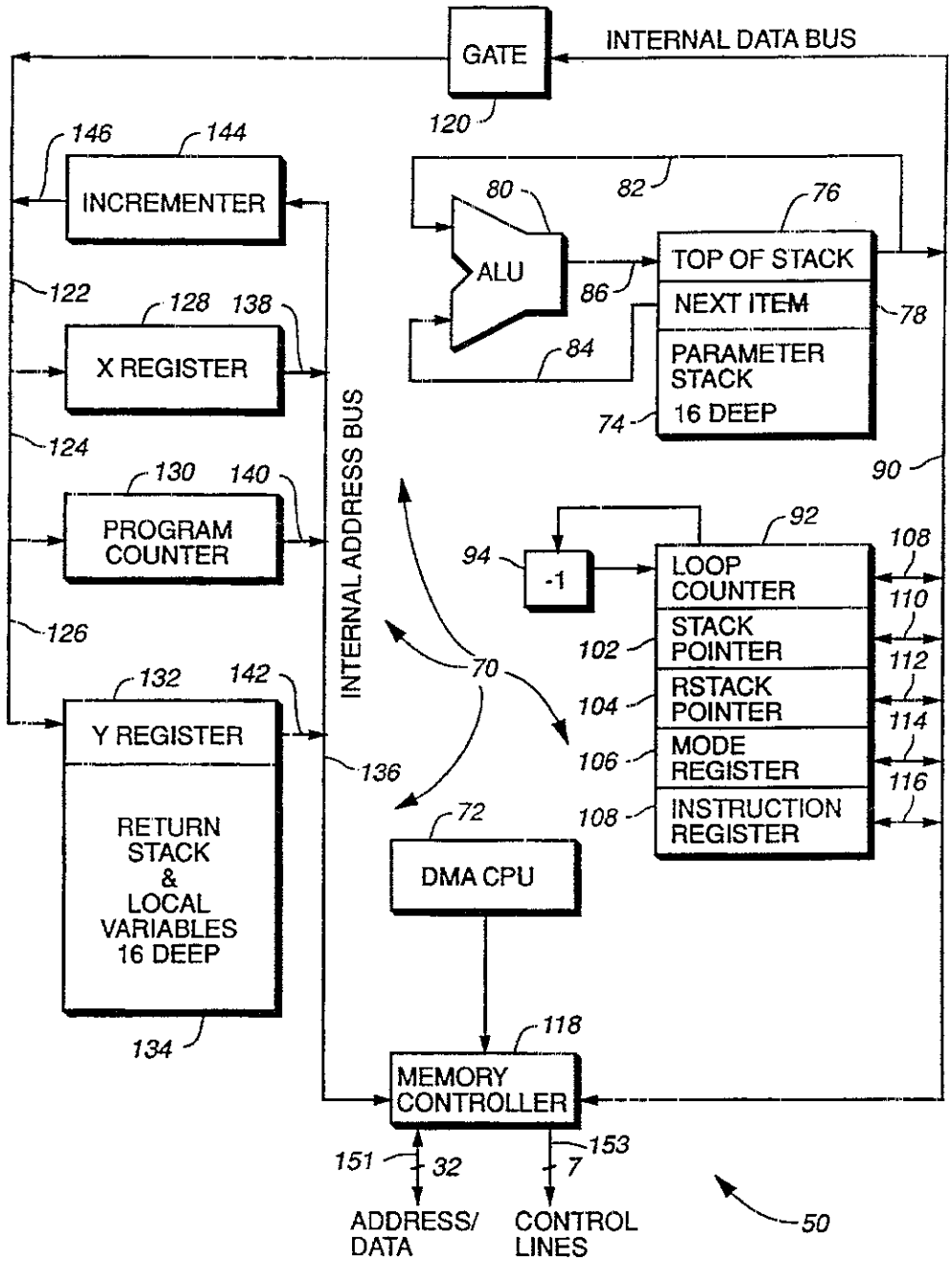

GATE
INTERNAL DATA BUS
120
146
144
INCREMENTER
80
82
76
ALU
TOP OF STACK
NEXT ITEM
PARAMETER STACK
16 DEEP
86
84
78
74
122
128
138
X REGISTER
124
130
140
PROGRAM COUNTER
126
INTERNAL ADDRESS BUS
90
92
94
-1
LOOP COUNTER
STACK POINTER
RSTACK POINTER
MODE REGISTER
INSTRUCTION REGISTER
108
110
112
114
116
102
104
106
108
70
132
142
Y REGISTER
RETURN STACK & LOCAL VARIABLES 16 DEEP
134
136
72
DMA CPU
118
MEMORY CONTROLLER
151
32
153
7
ADDRESS/ DATA
CONTROL LINES
50

FIG._2

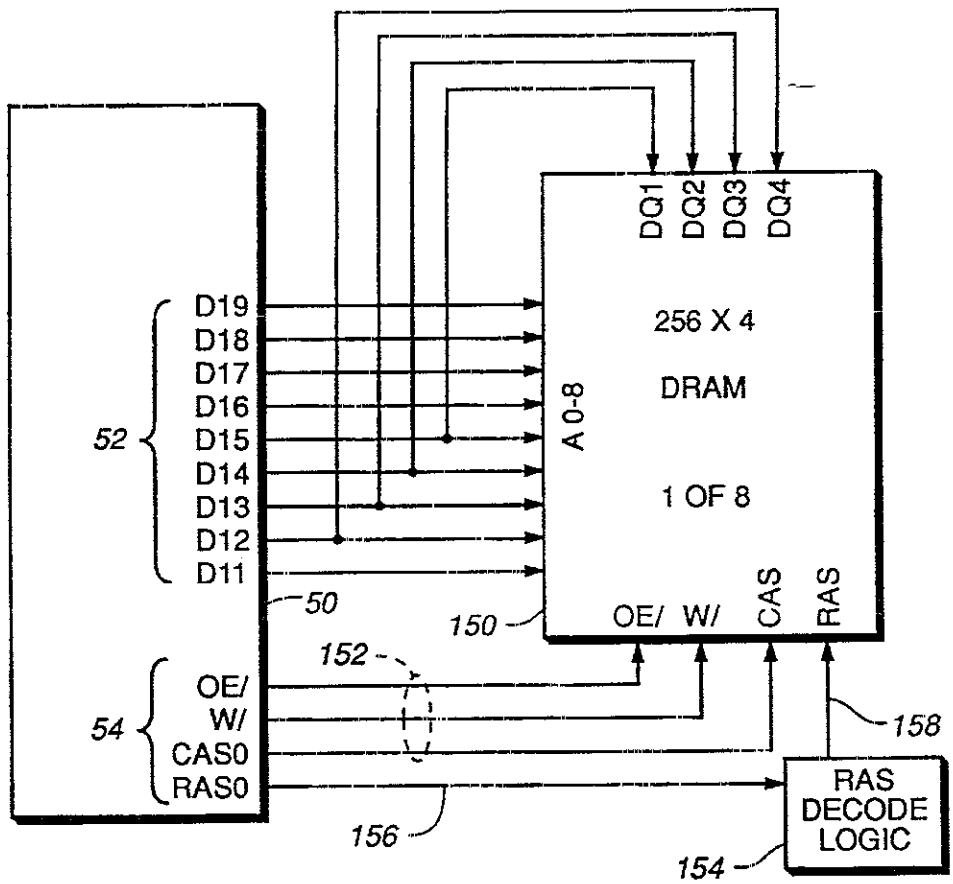

DQ1
DQ2
DQ3
DQ4
256 X 4
DRAM
1 OF 8
A 0-8
OE/
W/
CAS
RAS
D19
D18
D17
D16
D15
D14
D13
D12
D11
52
50
150
152
OE/
W/
CAS0
RAS0
54
156
158
RAS
DECODE
LOGIC
154

FIG._3

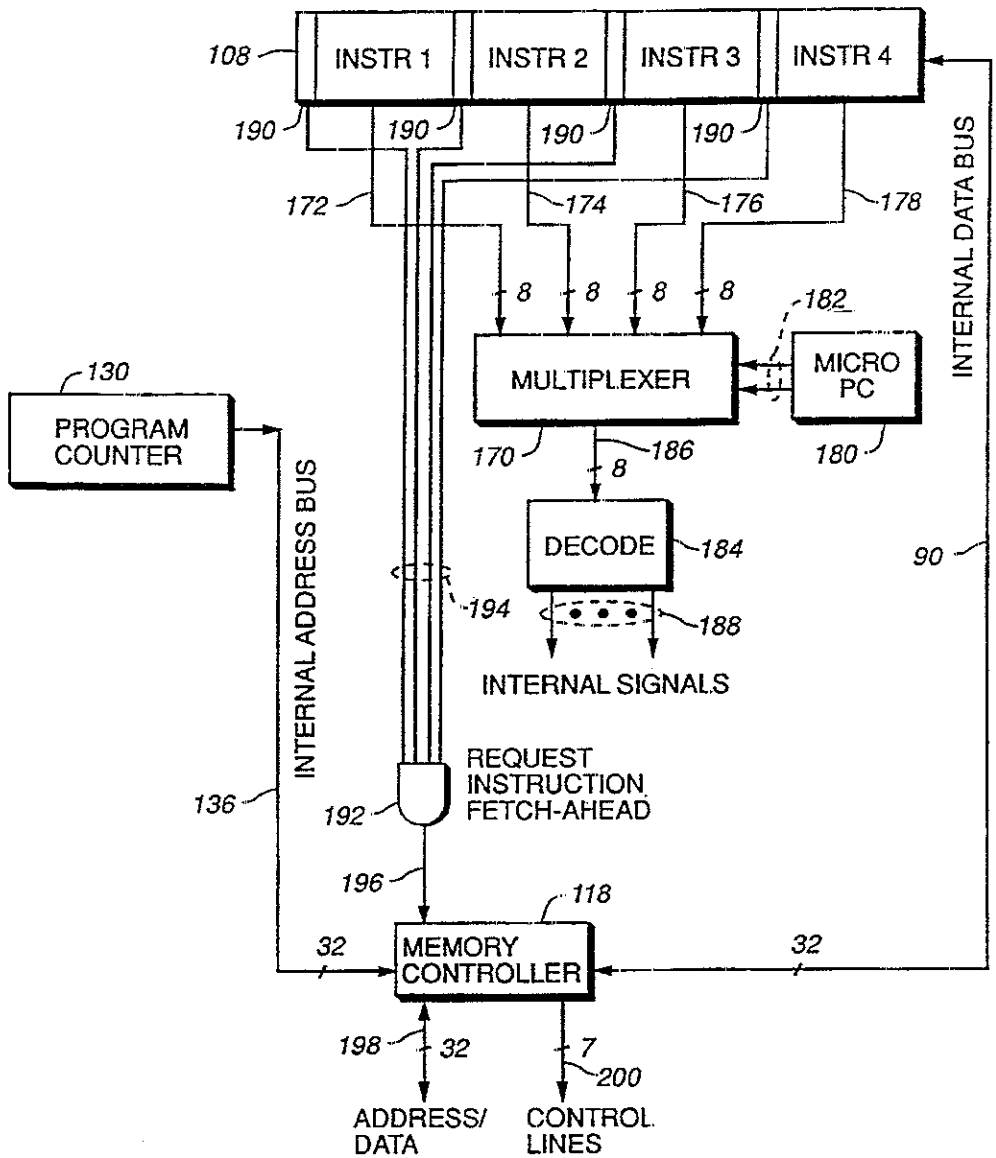

108
INSTR 1
INSTR 2
INSTR 3
INSTR 4
190
190
190
190
172
174
176
178
8
8
8
8
182
MULTIPLEXER
MICRO PC
INTERNAL DATA BUS
130
PROGRAM COUNTER
170
186
180
8
DECODE
184
90
194
188
INTERNAL SIGNALS
INTERNAL ADDRESS BUS
REQUEST INSTRUCTION FETCH-AHEAD
136
192
196
118
32
MEMORY CONTROLLER
32
198
32
7
200
ADDRESS/ DATA
CONTROL LINES

FIG._4

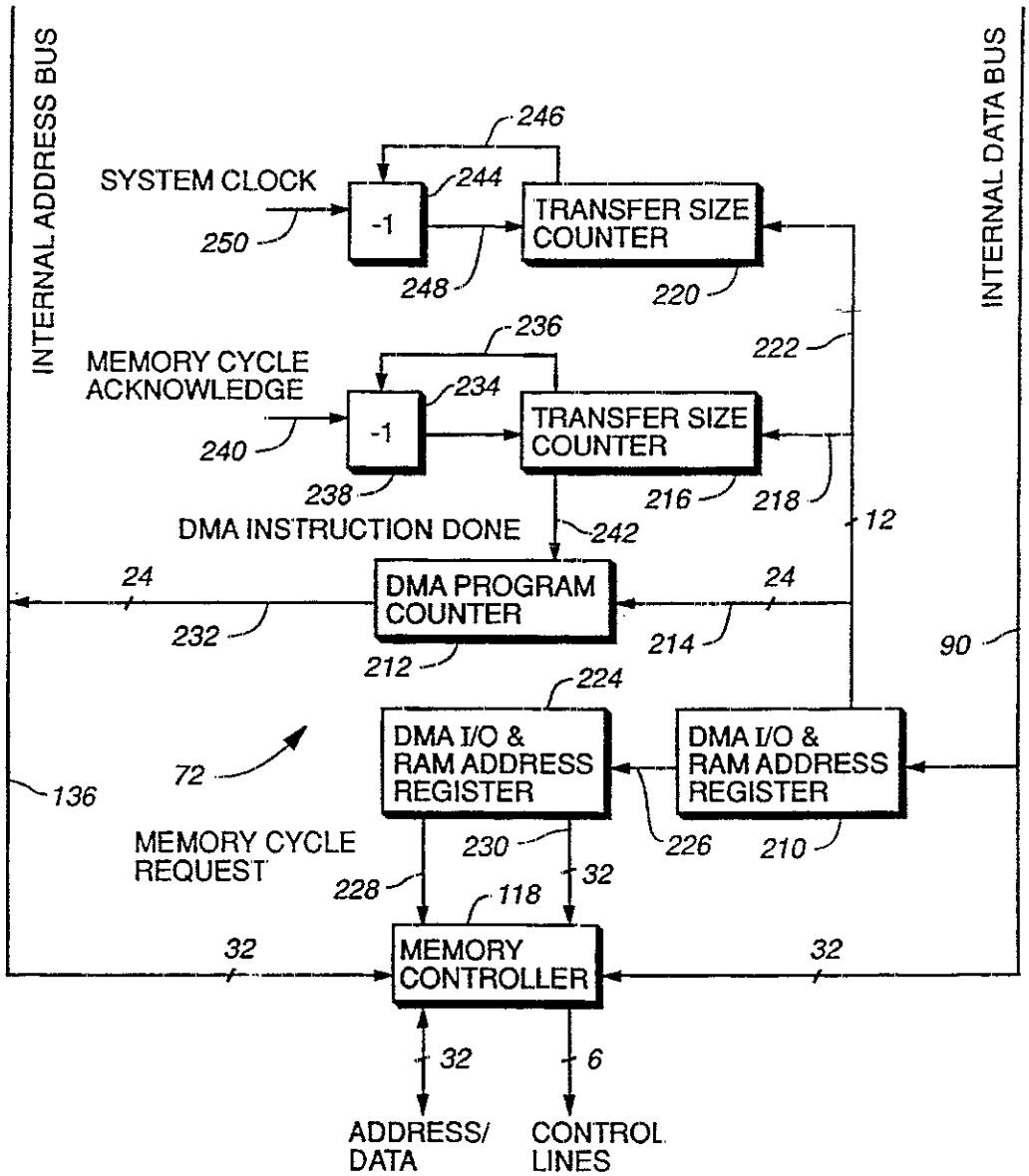

INTERNAL ADDRESS BUS
INTERNAL DATA BUS
SYSTEM CLOCK
250
-1
244
246
248
TRANSFER SIZE COUNTER
220
222
MEMORY CYCLE ACKNOWLEDGE
240
-1
234
236
238
TRANSFER SIZE COUNTER
216
218
12
DMA INSTRUCTION DONE
242
24
232
DMA PROGRAM COUNTER
212
24
214
90
224
72
136
DMA I/O & RAM ADDRESS REGISTER
DMA I/O & RAM ADDRESS REGISTER
226
210
MEMORY CYCLE REQUEST
230
228
32
118
32
MEMORY CONTROLLER
32
32
6
ADDRESS/ DATA
CONTROL LINES

FIG._5

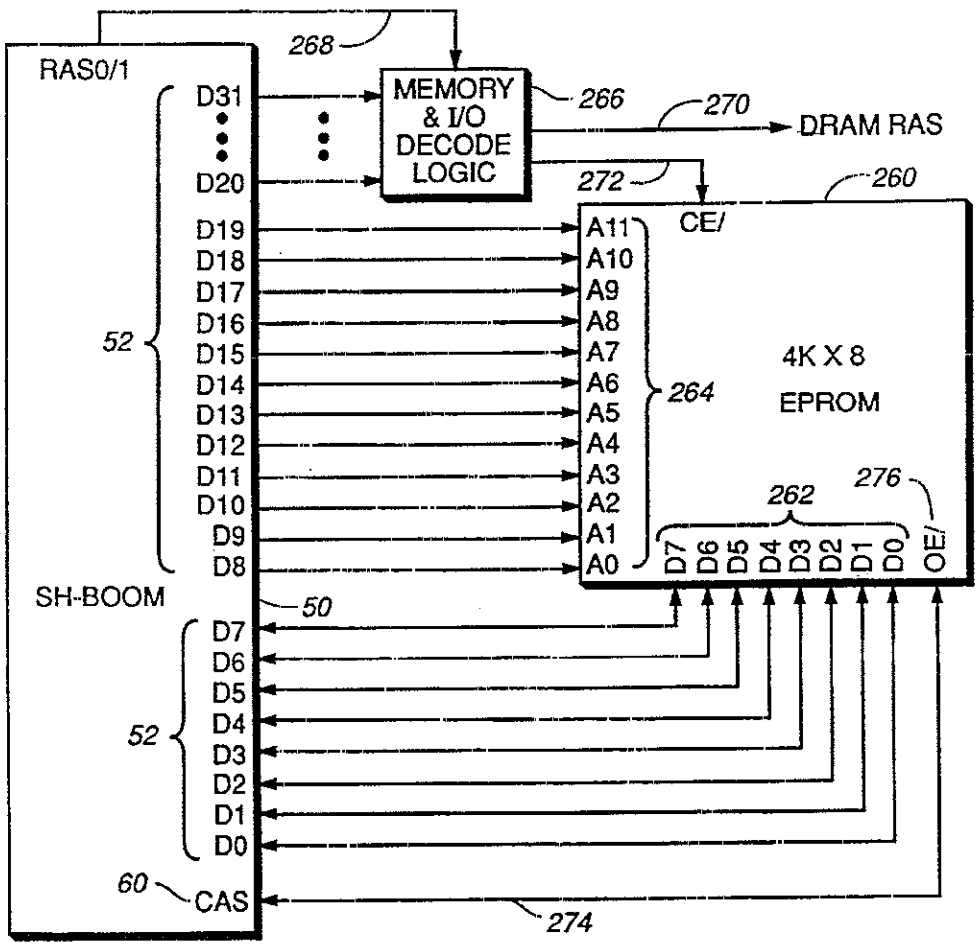

RAS0/1
268
MEMORY & I/O DECODE LOGIC
266
270
DRAM RAS
272
260
CE/
D31
D20
D19
D18
D17
D16
D15
D14
D13
D12
D11
D10
D9
D8
52
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
264
4K X 8
EPROM
262
276
D7
D6
D5
D4
D3
D2
D1
D0
OE/
SH-BOOM
50
D7
D6
D5
D4
D3
D2
D1
D0
52
60
CAS
274

FIG._6

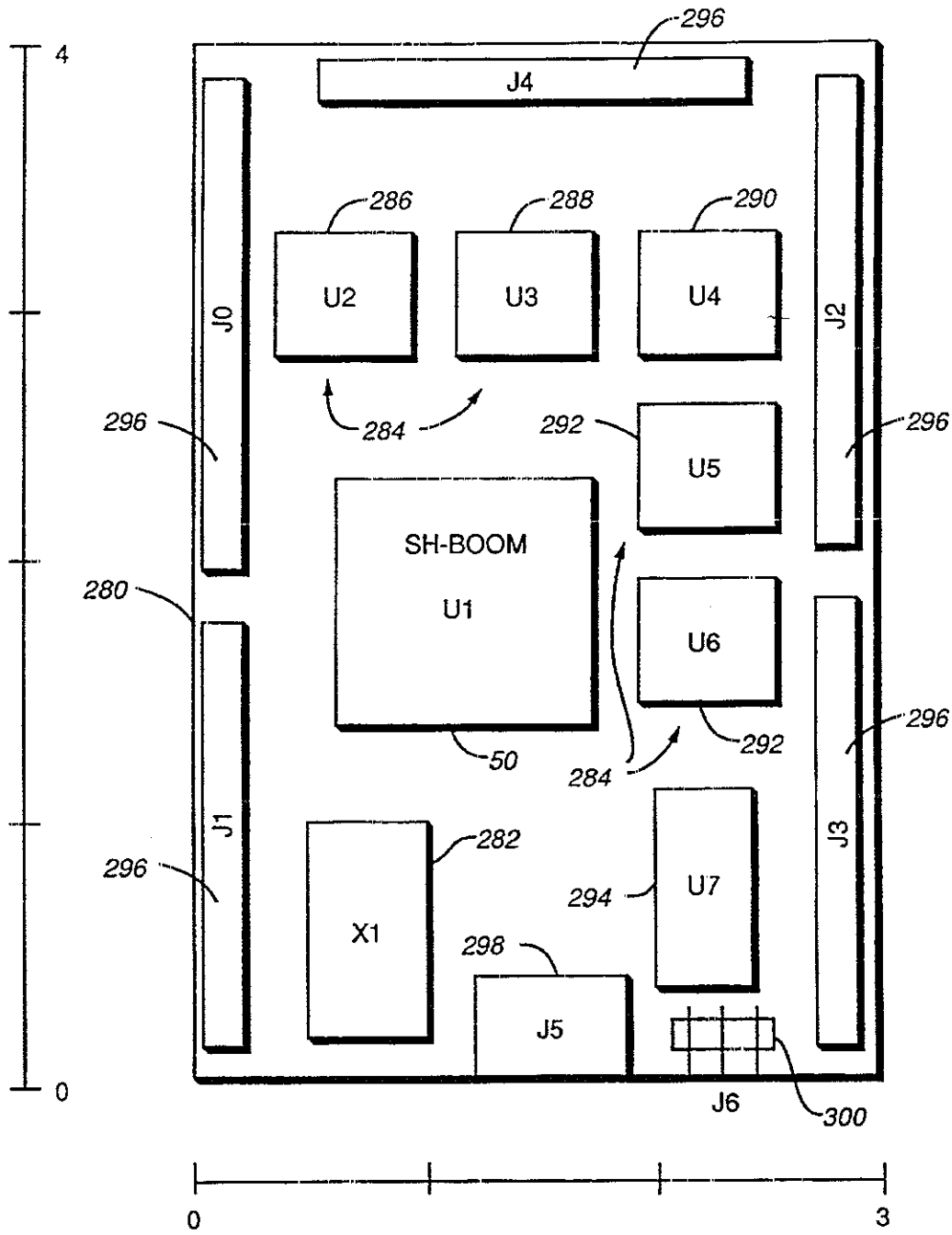

296
4
J4
286
288
290
U2
U3
U4
J0
J2
296
284
292
296
U5
SH-BOOM
U1
280
U6
296
50
284
292
J1
J3
282
296
294
U7
X1
298
J5
J6
300
0
0
3

FIG._7

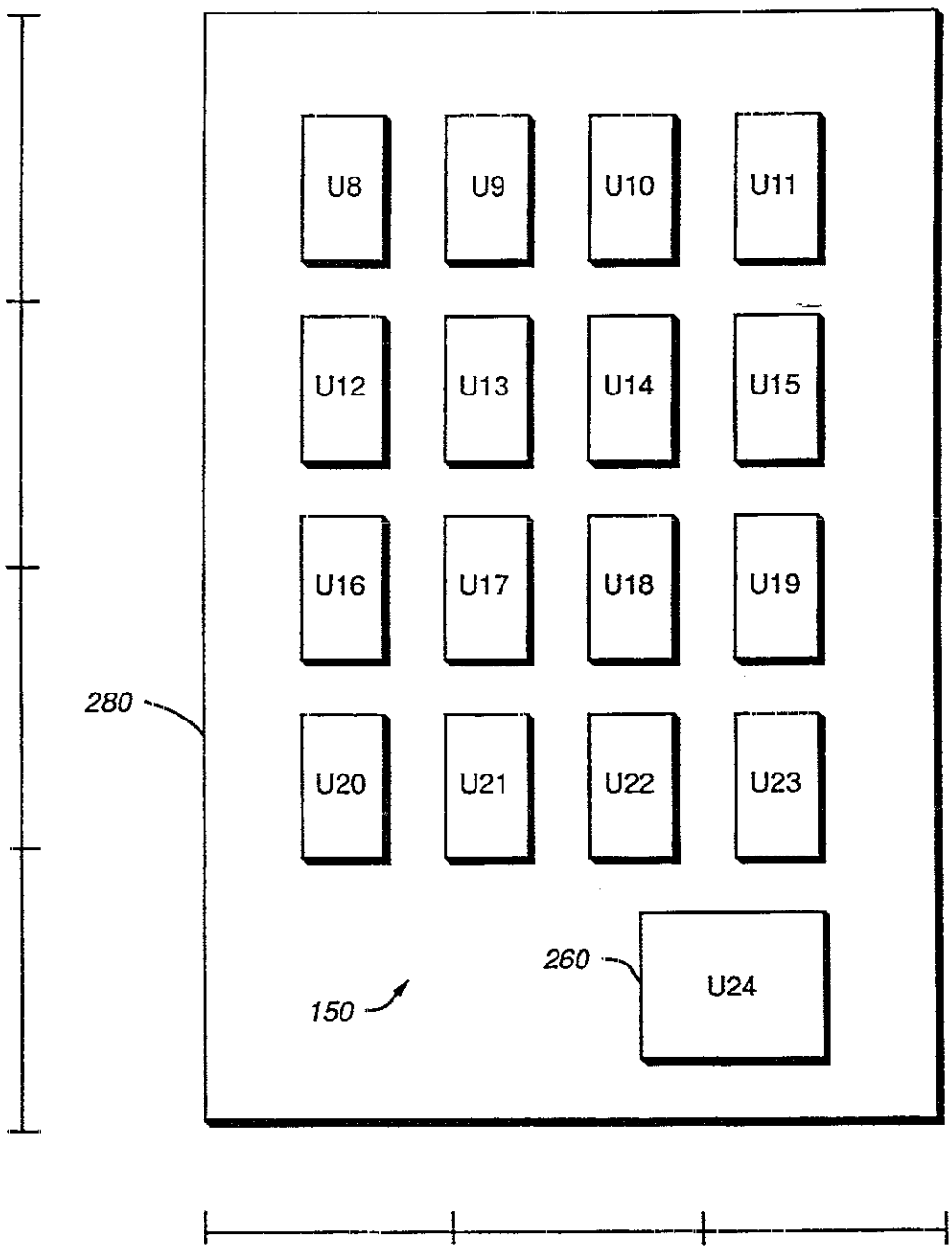

U8
U9
U10
U11
U12
U13
U14
U15
U16
U17
U18
U19
280
U20
U21
U22
U23
260
U24
150

FIG._8

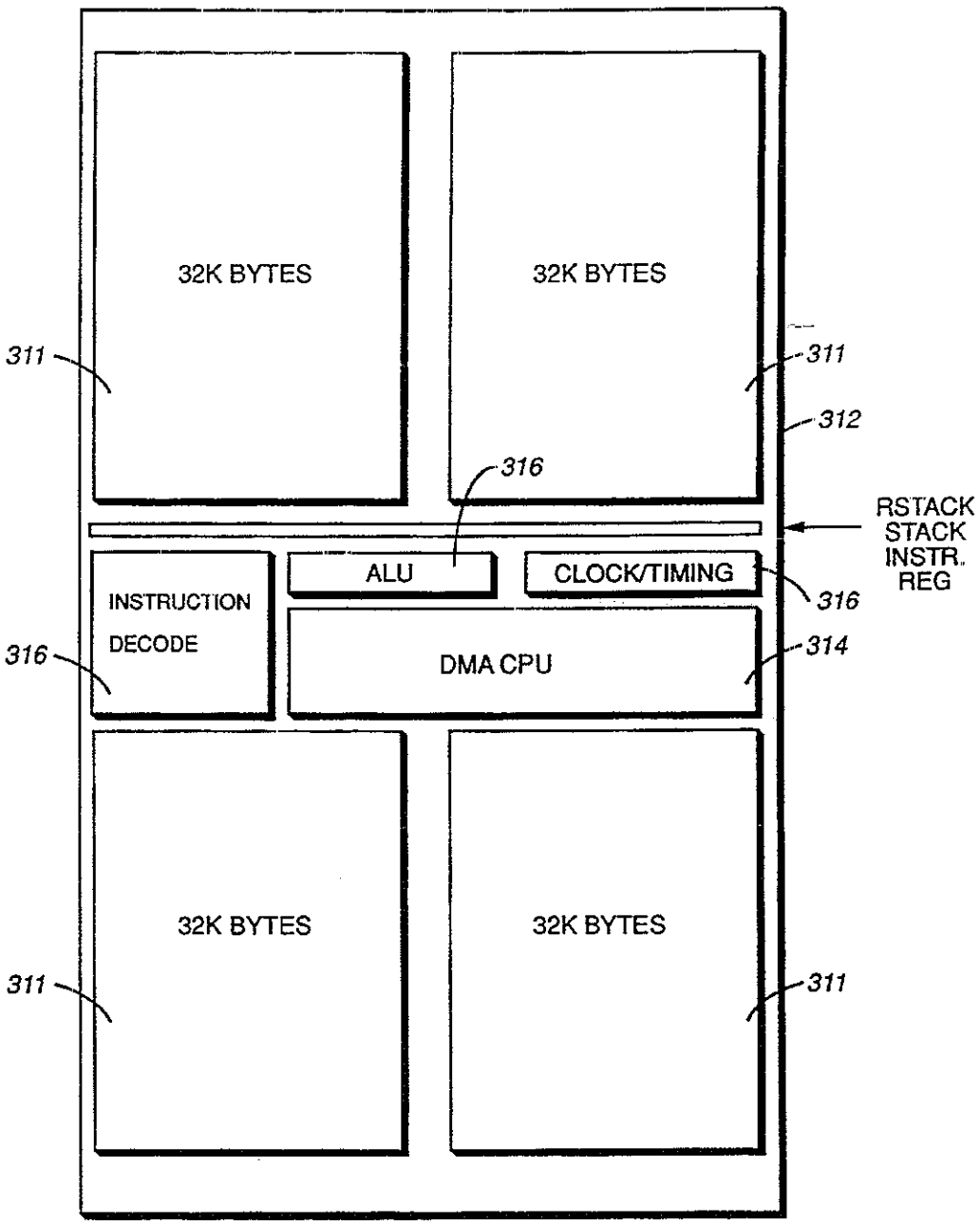

32K BYTES
32K BYTES
311
311
312
316
RSTACK
STACK
INSTR.
REG
ALU
CLOCK/TIMING
316
INSTRUCTION
DECODE
316
DMA CPU
314
32K BYTES
32K BYTES
311
311

FIG._9

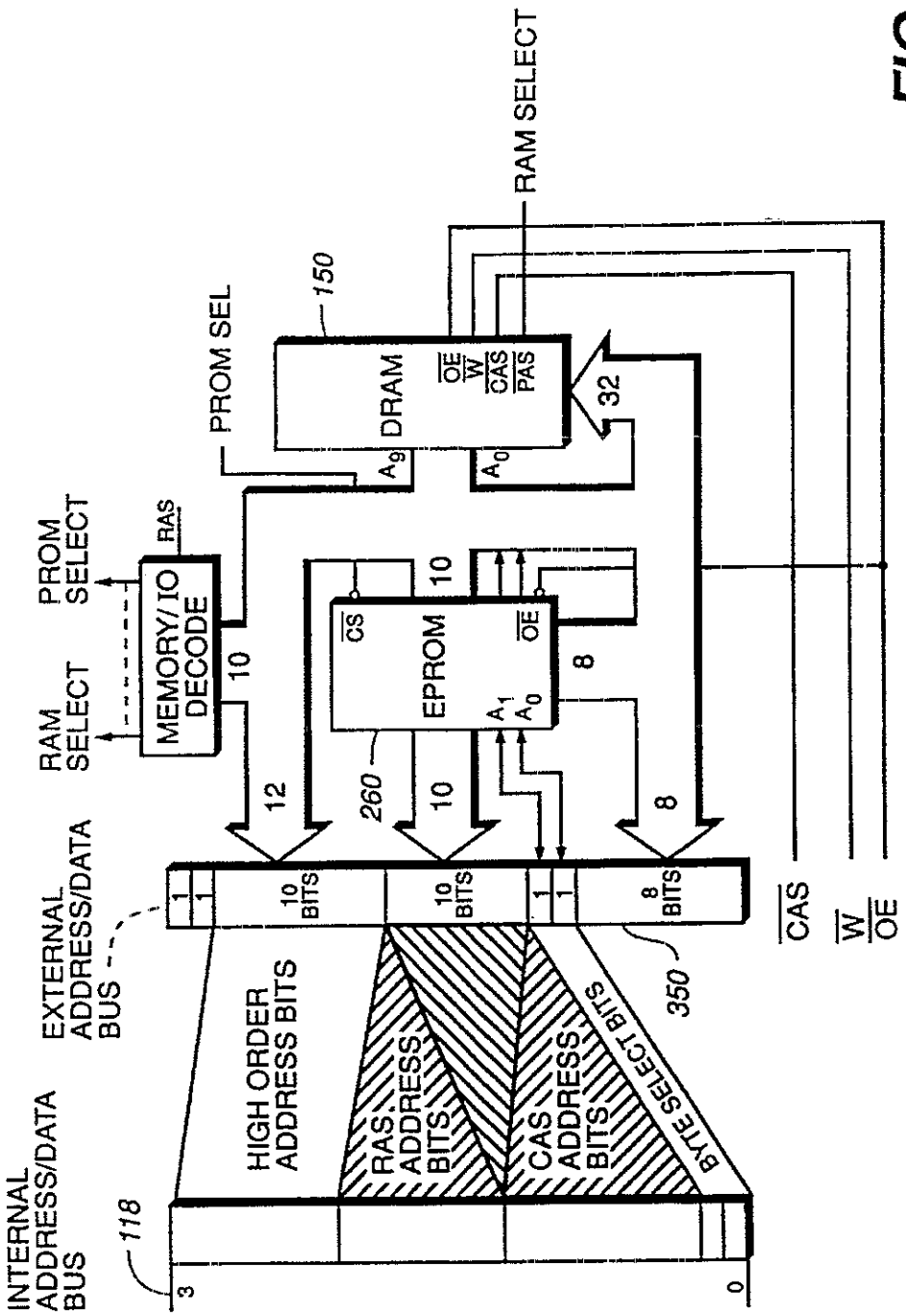
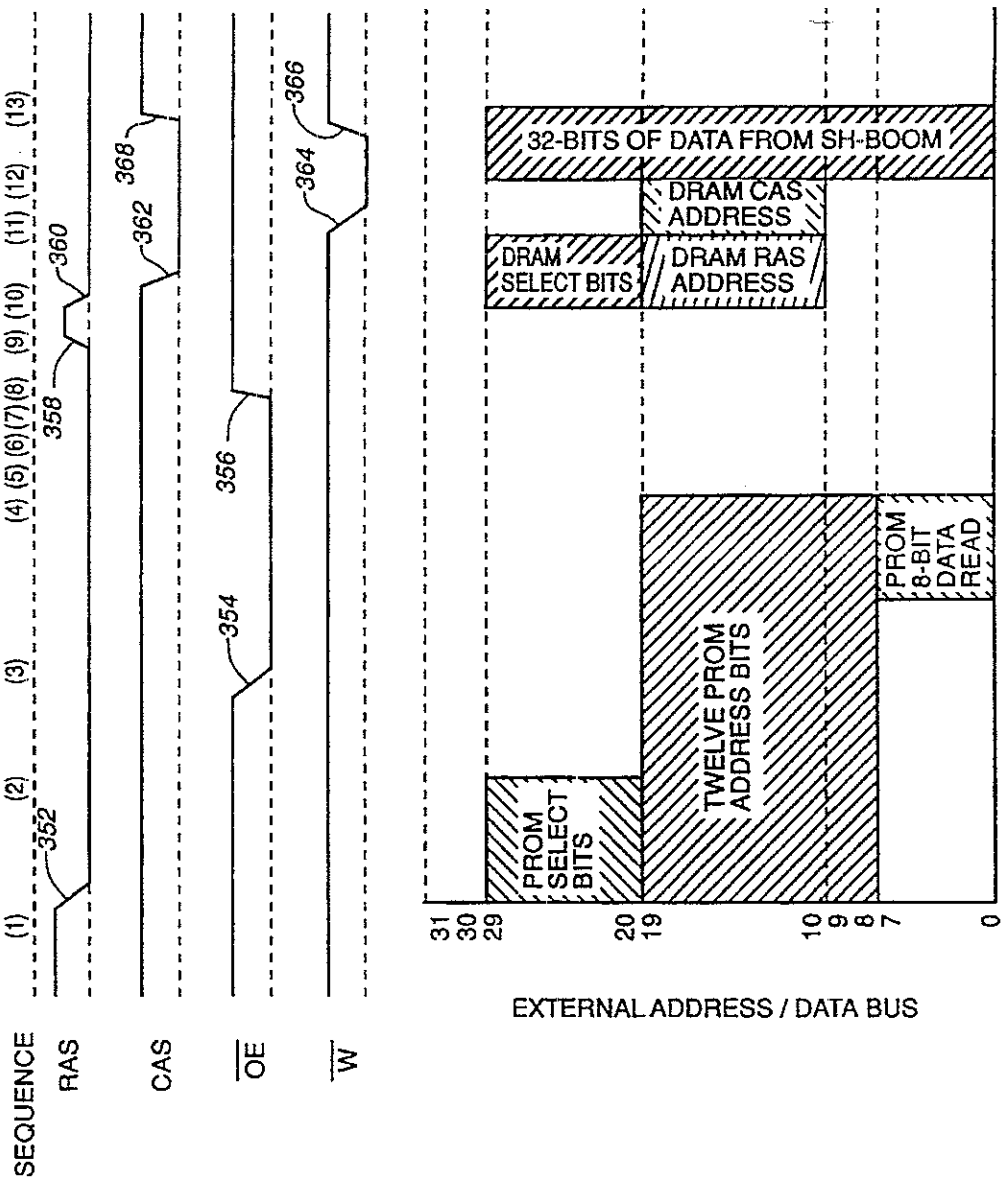

RAM SELECT
PROM SEL
150
DRAM
OE
W
CAS
RAS
32
A9
A0
PROM SELECT
RAS
MEMORY/IO DECODE
10
RAM SELECT
CS
EPROM
OE
A1
A0
10
8
260
12
10
8
EXTERNAL ADDRESS/DATA BUS
1
1
10 BITS
10 BITS
1
1
8 BITS
CAS
W
OE
350
HIGH ORDER ADDRESS BITS
RAS ADDRESS BITS
CAS ADDRESS BITS
BYTE SELECT BITS
INTERNAL ADDRESS/DATA BUS
118
3
0

FIG._10

FIG._11


SEQUENCE
RAS
CAS
OE
W
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
(12)
(13)
352
354
356
358
360
362
364
366
368
32-BITS OF DATA FROM SH-BOOM
DRAM CAS ADDRESS
DRAM SELECT BITS
DRAM RAS ADDRESS
PROM SELECT BITS
TWELVE PROM ADDRESS BITS
PROM 8-BIT DATA READ
31
30
29
20
19
10
9
8
7
0
EXTERNAL ADDRESS / DATA BUS

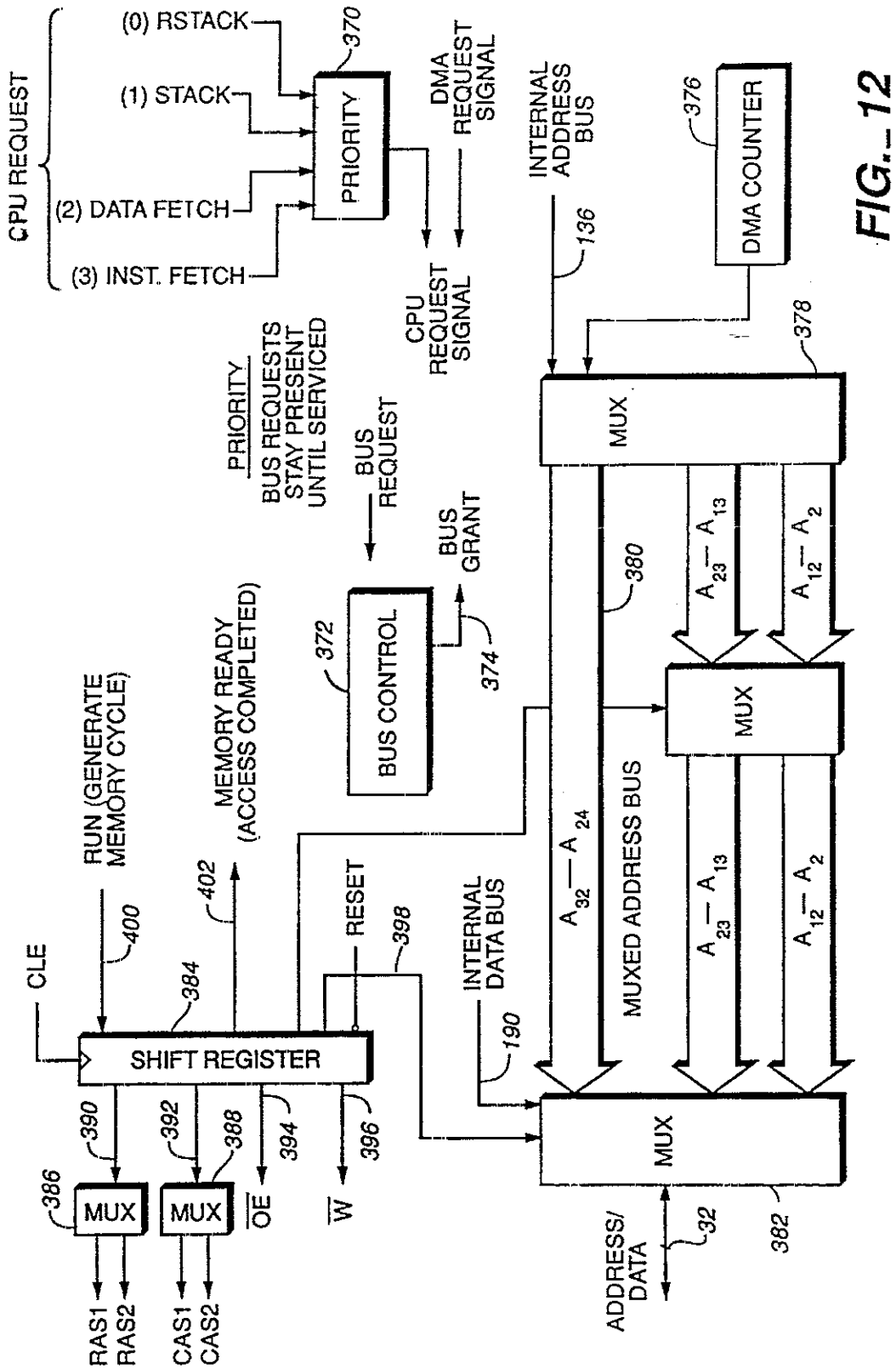

CPU REQUEST
(0) RSTACK
(1) STACK
(2) DATA FETCH
(3) INST. FETCH
370
PRIORITY
DMA REQUEST SIGNAL
CPU REQUEST SIGNAL
INTERNAL ADDRESS BUS
136
376
DMA COUNTER
PRIORITY
BUS REQUESTS STAY PRESENT UNTIL SERVICED
BUS REQUEST
BUS GRANT
372
BUS CONTROL
374
MEMORY READY (ACCESS COMPLETED)
RUN (GENERATE MEMORY CYCLE)
400
402
CLE
384
SHIFT REGISTER
RESET
398
INTERNAL DATA BUS
190
MUXED ADDRESS BUS
378
MUX
380
A23 — A13
A12 — A2
A32 — A24
MUX
MUX
MUX
386
390
392
388
394
396
OE
W
RAS1
RAS2
CAS1
CAS2
ADDRESS/ DATA
32
382

FIG._12

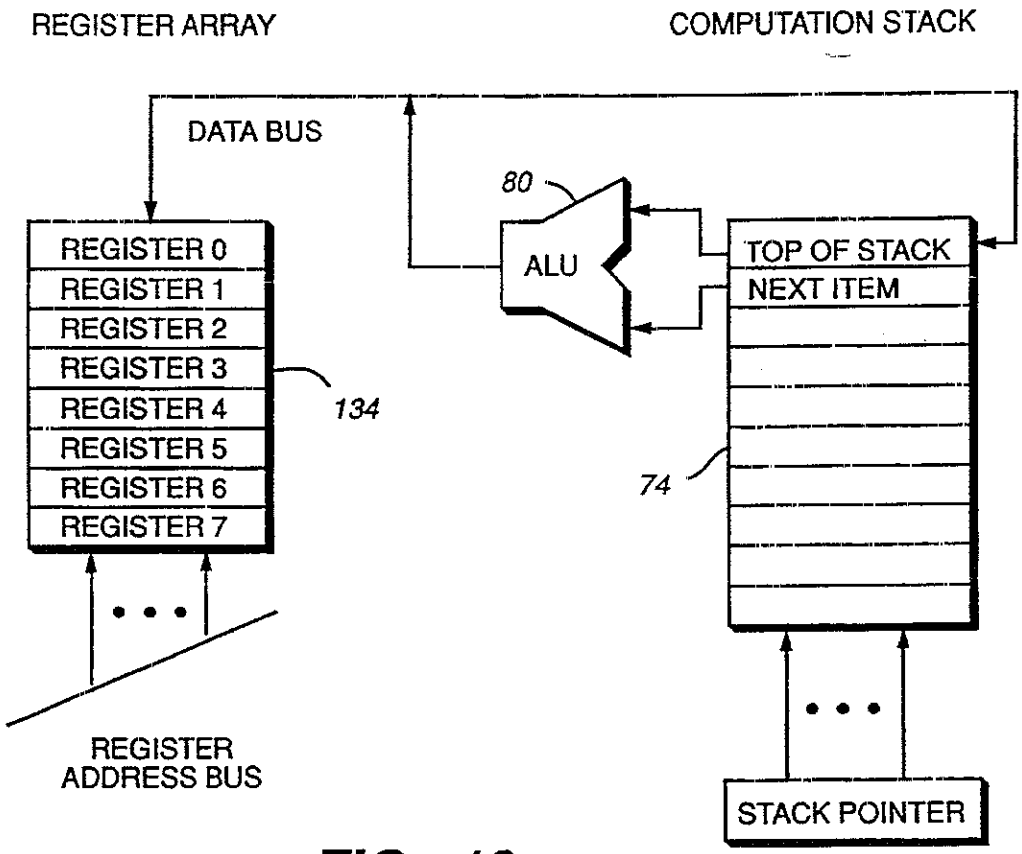

REGISTER ARRAY
COMPUTATION STACK
DATA BUS
80
ALU
REGISTER 0
REGISTER 1
REGISTER 2
REGISTER 3
REGISTER 4
REGISTER 5
REGISTER 6
REGISTER 7
134
TOP OF STACK
NEXT ITEM
74
REGISTER
ADDRESS BUS
STACK POINTER

FIG._13

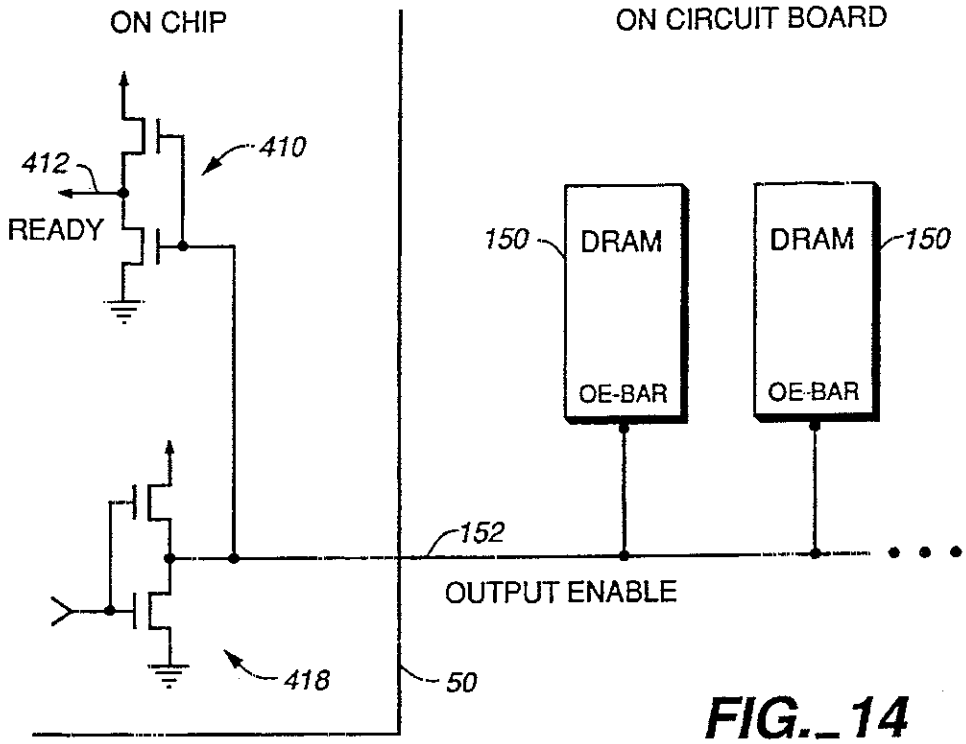

ON CHIP
ON CIRCUIT BOARD
412
410
READY
150
DRAM
OE-BAR
DRAM
150
OE-BAR
152
OUTPUT ENABLE
418
50

FIG._14

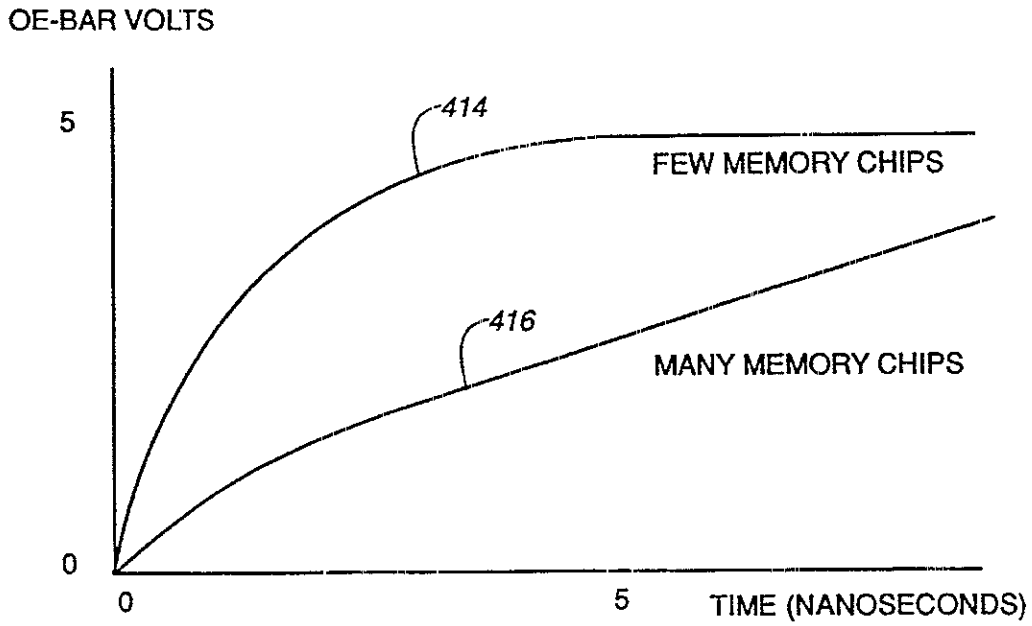

OE-BAR VOLTS
5
414
FEW MEMORY CHIPS
416
MANY MEMORY CHIPS
0
0
5
TIME (NANOSECONDS)

FIG._15

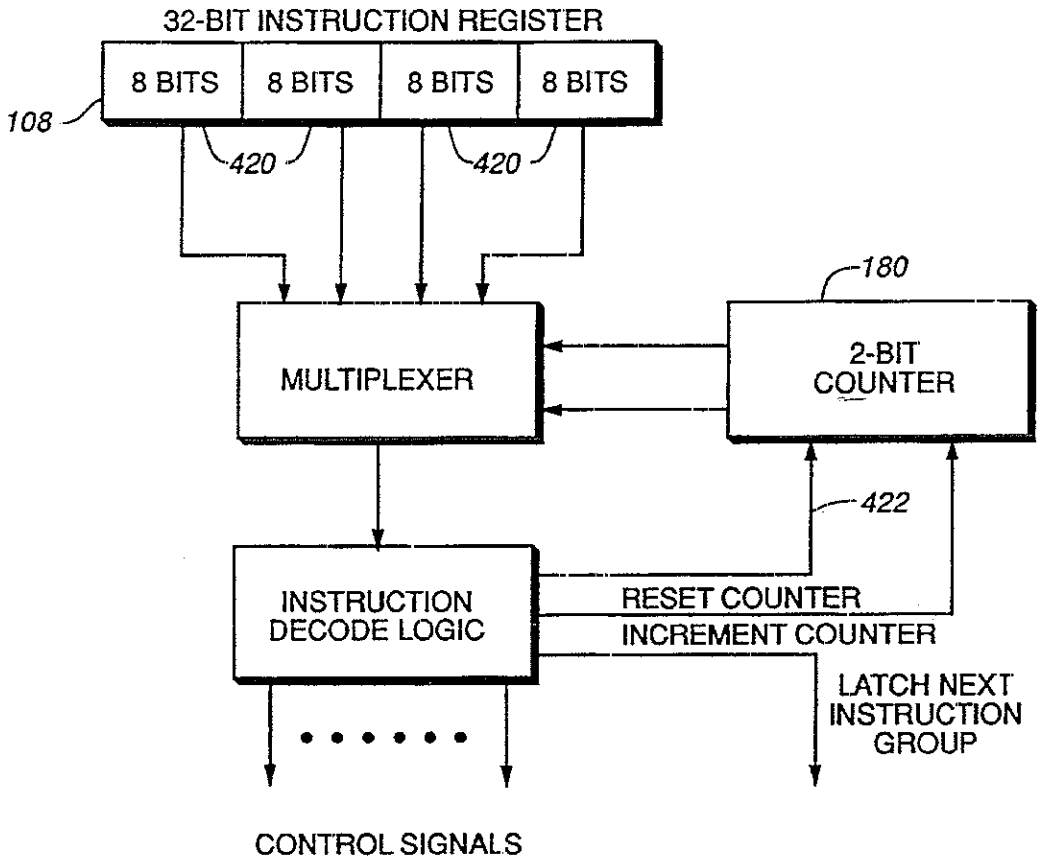

32-BIT INSTRUCTION REGISTER
8 BITS
8 BITS
8 BITS
8 BITS
108
420
420
180
MULTIPLEXER
2-BIT COUNTER
422
INSTRUCTION DECODE LOGIC
RESET COUNTER
INCREMENT COUNTER
LATCH NEXT INSTRUCTION GROUP
CONTROL SIGNALS

FIG._16

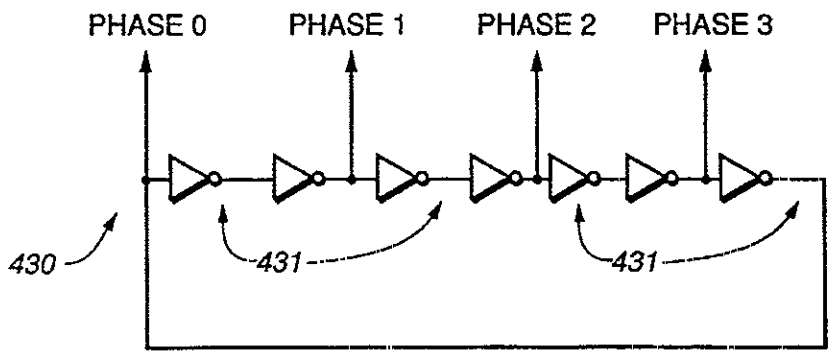

PHASE 0
PHASE 1
PHASE 2
PHASE 3
430
431
431

FIG._18

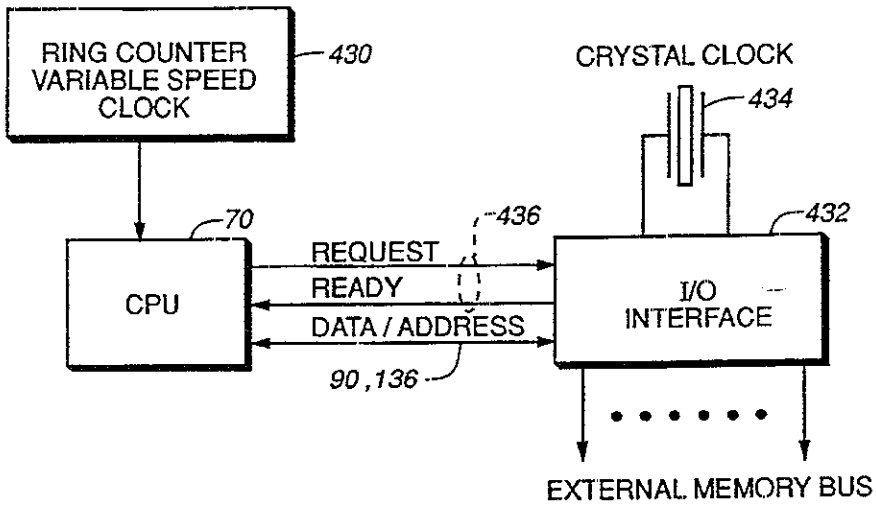

RING COUNTER VARIABLE SPEED CLOCK
430
CRYSTAL CLOCK
434
70
436
432
CPU
REQUEST
READY
DATA / ADDRESS
I/O INTERFACE
90 ,136
EXTERNAL MEMORY BUS

FIG._17

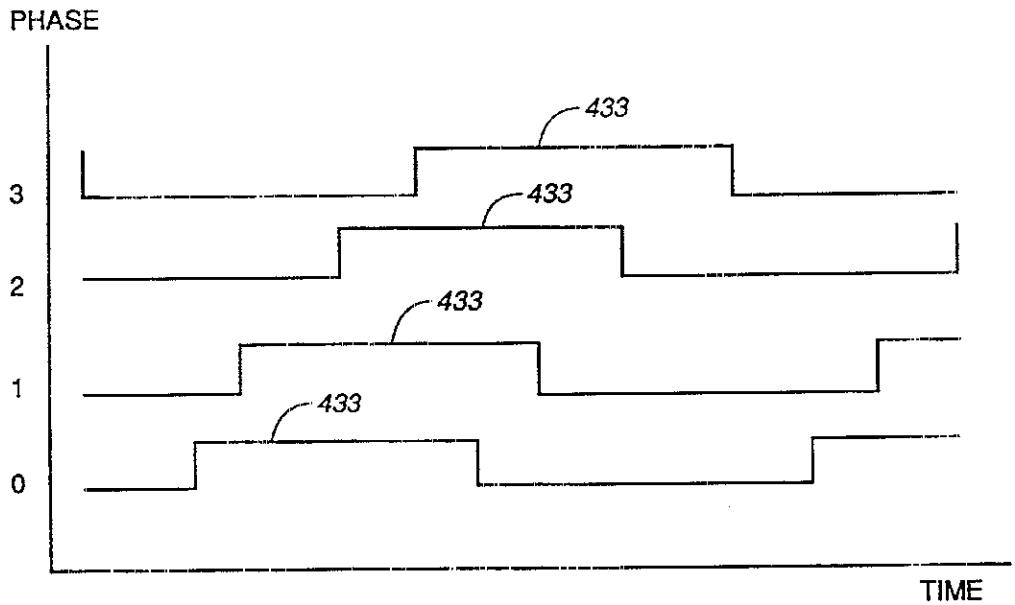

PHASE
433
433
433
433
3
2
1
0
TIME

FIG._19

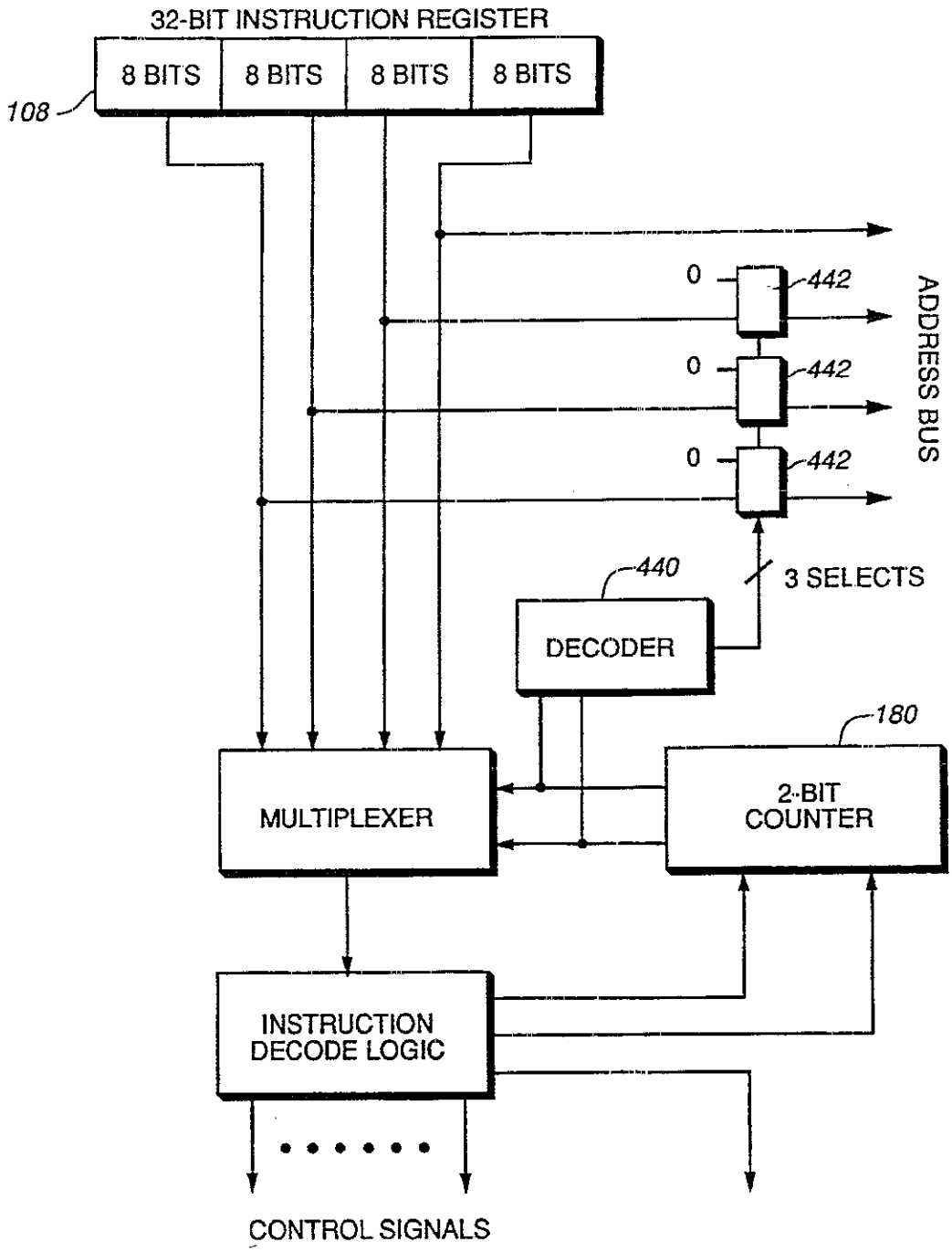

32-BIT INSTRUCTION REGISTER
8 BITS
8 BITS
8 BITS
8 BITS
108
0
442
0
442
0
442
ADDRESS BUS
440
3 SELECTS
DECODER
180
MULTIPLEXER
2-BIT COUNTER
INSTRUCTION DECODE LOGIC
CONTROL SIGNALS

FIG._20

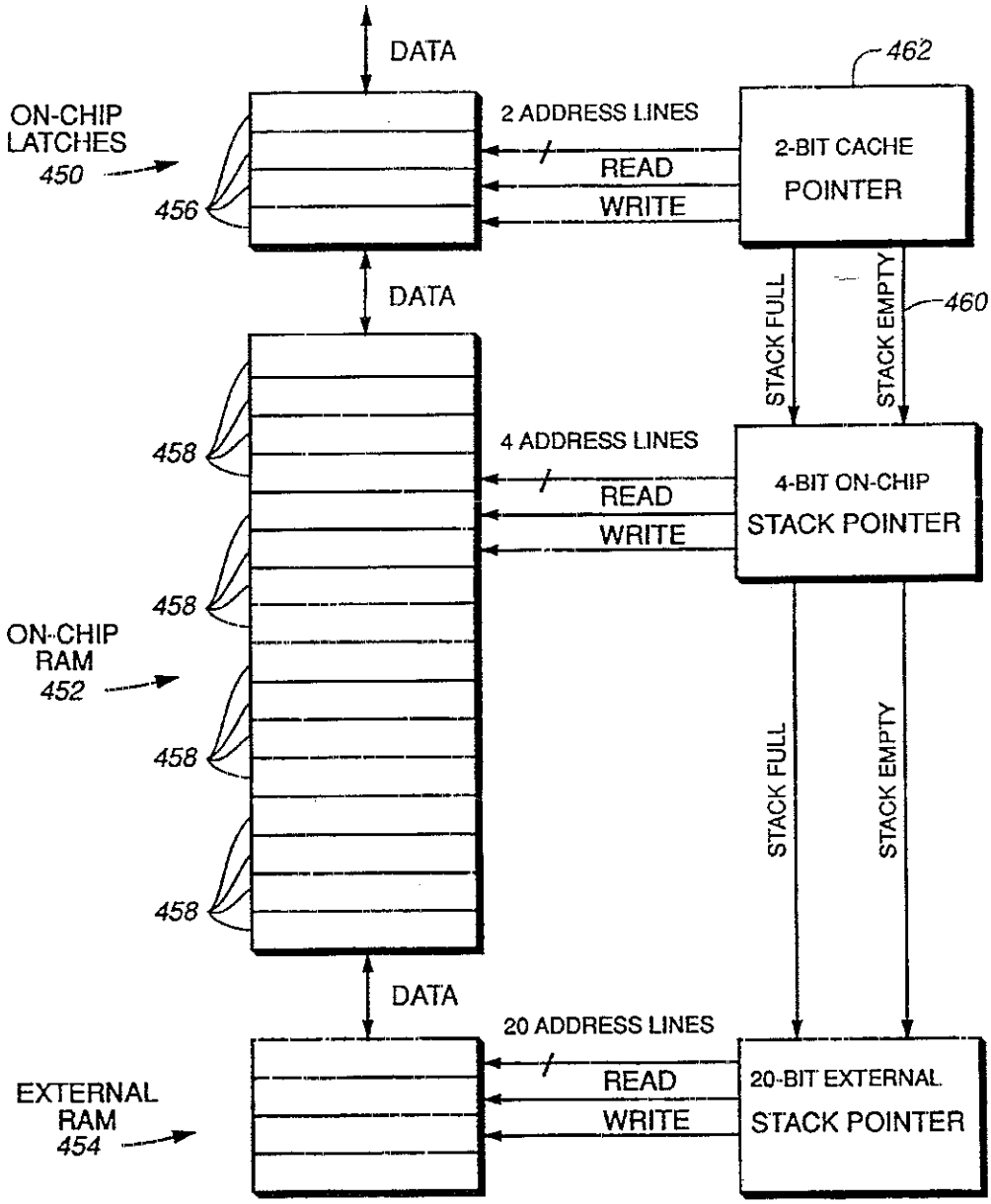

DATA
462
ON-CHIP LATCHES
450
2 ADDRESS LINES
READ
WRITE
2-BIT CACHE POINTER
456
DATA
STACK FULL
STACK EMPTY
460
458
4 ADDRESS LINES
READ
WRITE
4-BIT ON-CHIP STACK POINTER
458
ON-CHIP RAM
452
458
STACK FULL
STACK EMPTY
458
DATA
20 ADDRESS LINES
READ
WRITE
20-BIT EXTERNAL STACK POINTER
EXTERNAL RAM
454

FIG._21

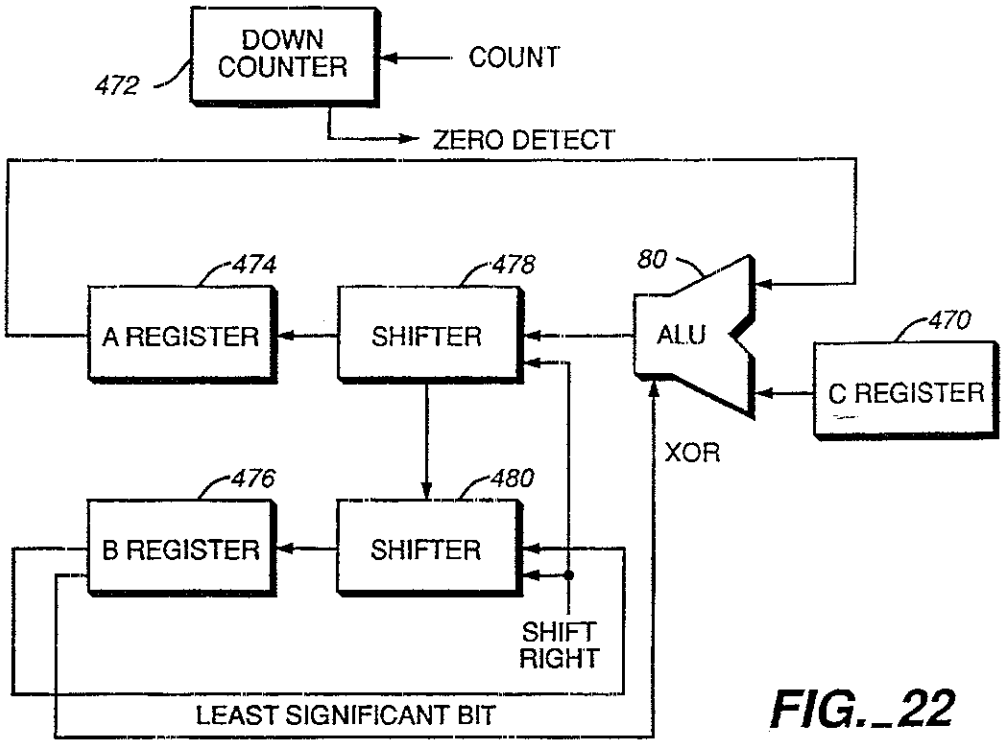

DOWN COUNTER
472
COUNT
ZERO DETECT
474
A REGISTER
478
SHIFTER
80
ALU
470
C REGISTER
XOR
476
B REGISTER
480
SHIFTER
SHIFT RIGHT
LEAST SIGNIFICANT BIT

FIG._22

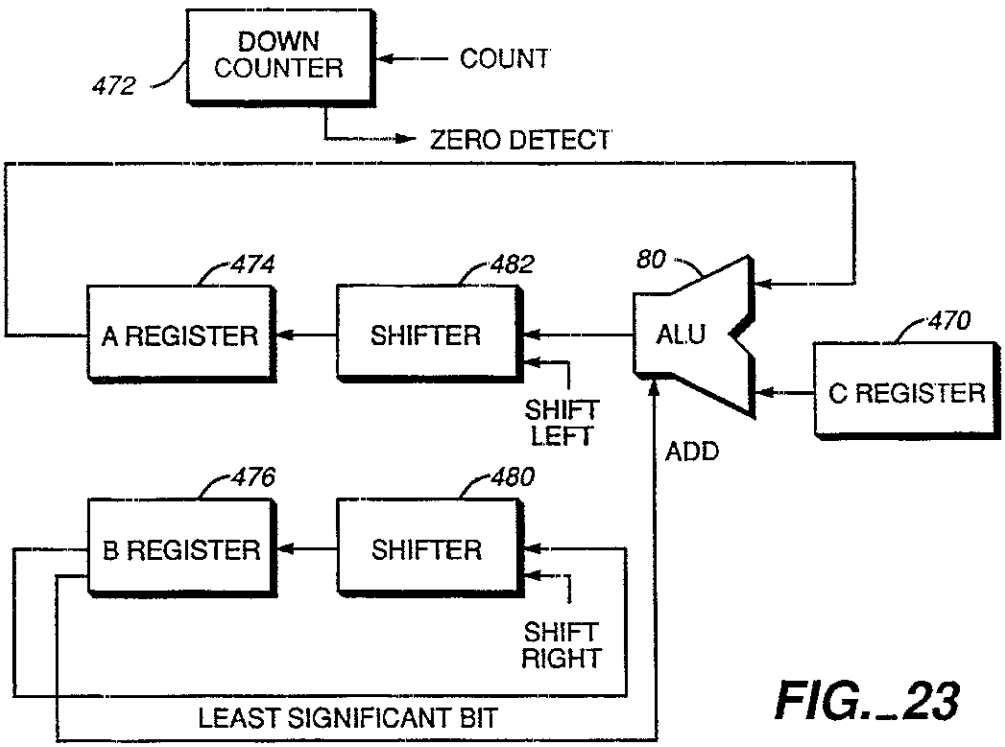

DOWN COUNTER
472
COUNT
ZERO DETECT
474
A REGISTER
482
SHIFTER
80
ALU
470
C REGISTER
SHIFT LEFT
ADD
476
B REGISTER
480
SHIFTER
SHIFT RIGHT
LEAST SIGNIFICANT BIT

FIG._23

HIGH PERFORMANCE, LOW COSI MICROPROCESSOR ARCHITECTURE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to a simplified, reduced instruction set computer (RISC) microprocessor. More particularly, it relates to such a microprocessor which is capable of performance levels of, for example, 20 million instructions per second (MIPS) at a price of, for example, 20 dollars.

2. Description of the Prior Art

Since the invention of the microprocessor, improvements in its design have taken two different approaches. In the first approach, a brute force gain in performance has been achieved through the provision of greater numbers of faster transistors in the microprocessor integrated circuit and an instruction set of increased complexity. This approach is exemplified by the Motorola 68000 and Intel 80X86 microprocessor families. The trend in this approach is to larger die sizes and packages, with hundreds of pinouts.

More recently, it has been perceived that performance gains can be achieved through comparative simplicity, both in the microprocessor integrated circuit itself and in its instruction set. This second approach provides RISC microprocessors, and is exemplified by the Sun SPARC and The Intel 8960 microprocessors However, even with this approach as conventionally practiced, the packages for the microprocessor are large, in order to accommodate the large number of pinouts that continue to be employed. A need therefore remains for further simplification of high performance microprocessors.

With conventional high performance microprocessors, fast static memories are required for direct connection to the microprocessors in order to allow memory accesses that are fast enough to keep up with the microprocessors Slower dynamic random access memories (DRAMs) are used with such microprocessors only in a hierarchical memory arrangement, with the static memories acting as a buffer between the microprocessors and the DRAMs. The necessity to use static memories increases cost of the resulting systems.

Conventional microprocessors provide direct memory accesses (DMA) for system peripheral units through DMA controllers, which may be located on the microprocessor integrated circuit, or provided separately. Such DMA controllers can provide routine handling of DMA requests and responses, but some processing by the main central processing unit (CPU) of the microprocessor is required

SUMMARY OF THE INVENTION

Accordingly, it is an object of this invention to provide a microprocessor with a reduced pin count and cost compared to conventional microprocessors.

It is another object of the invention to provide a high performance microprocessor that can be directly connected to DRAMs without sacrificing microprocessor speed.

It is a further object of the invention to provide a high performance microprocessor in which DMA does not require use of the main CPU during DMA requests and responses and which provides very rapid DMA response with predictable response times.

The attainment of these and related objects may be achieved through use of the novel high performance, low cost microprocessor herein disclosed In accordance with one aspect of the invention, a microprocessor system in accordance with this invention has a central processing unit, a dynamic random access memory and a bus connecting the central processing unit to the dynamic random access memory. There is a multiplexing means on the bus between the central processing unit and the dynamic random access memory. The multiplexing means is connected and configured to provide row addresses, column addresses and data on the bus.

In accordance with another aspect of the invention, the microprocessor system has a means connected to the bus for fetching instructions for the central processing unit on the bus The means for fetching instructions is configured to fetch multiple sequential instructions in a single memory cycle. In a variation of this aspect of the invention, a programmable read only memory containing instructions for the central processing unit is connected to the bus. The means for fetching instructions includes means for assembling a plurality of instructions from the programmable read only memory and storing the plurality of instructions in the dynamic random access memory.

In another aspect of the invention, the microprocessor system includes a central processing unit, a direct memory access processing unit and a memory connected by a bus. The direct memory access processing unit includes means for fetching instructions for the central processing unit and for fetching instructions for the direct memory access processing unit on the bus.

In a further aspect of the invention, the microprocessor system, including the memory, is contained in an integrated circuit The memory is a dynamic random access memory, and the means for fetching multiple instructions includes a column latch for receiving the multiple instructions

In still another aspect of the invention, the microprocessor system additionally includes an instruction register for the multiple instructions connected to the means for fetching instructions. A means is connected to the instruction register for supplying the multiple instructions in succession from the instruction register. A counter is connected to control the means for supplying the multiple instructions to supply the multiple instructions in succession A means for decoding the multiple instructions is connected to receive the multiple instructions in succession from the means for supplying the multiple instructions The counter is connected to said means for decoding to receive incrementing and reset control signals from the means for decoding. The means for decoding is configured to supply the reset control signal to the counter and to supply a control signal to the means for fetching instructions in response to a SKIP instruction in the multiple instructions. In a modification of this aspect of the invention, the microprocessor system additionally has a loop counter connected to receive a decrement control signal from the means for decoding. The means for decoding is configured to supply the reset control signal to the counter and the decrement control signal to the loop counter in response to a MICROLOOP instruction in the multiple instructions. In a further modification to this aspect of the invention, the means for decoding is configured to control the counter in response to an instruction utilizing a variable width operand. A means is connected to

the counter to select the variable width operand in response to the counter.

In a still further aspect of the invention, the microprocessor system includes an arithmetic logic unit. A first push down stack is connected to the arithmetic logic unit. The first push down stack includes means for storing a top item connected to a first input of the arithmetic logic unit and means for storing a next item connected to a second input of the arithmetic logic unit. The arithmetic logic unit has an output connected to the means for storing a top item. The means for storing a top item is connected to provide an input to a register file. The register file desirably is a second push down stack, and the means for storing a top item and the register file are bidirectionally connected.

In another aspect of the invention, a data processing system has a microprocessor including a sensing circuit and a driver circuit, a memory, and an output enable line connected between the memory, the sensing circuit and the driver circuit. The sensing circuit is configured to provide a ready signal when the output enable line reaches a predetermined electrical level, such as a voltage. The microprocessor is configured so that the driver circuit provides an enabling signal on the output enable line responsive to the ready signal.

In a further aspect of the invention, the microprocessor system has a ring counter variable speed system clock connected to the central processing unit. The central processing unit and the ring counter variable speed system clock are provided in a single integrated circuit. An input/output interface is connected to exchange coupling control signals, addresses and data with the input/output interface. A second clock independent of the ring counter variable speed system clock is connected to the input/output interface.

In yet another aspect of the invention, a push down stack is connected to the arithmetic logic unit. The push down stack includes means for storing a top item connected to a first input of the arithmetic logic unit and means for storing a next item connected to a second input of the arithmetic logic unit. The arithmetic logic unit has an output connected to the means for storing a top item. The push down stack has a first plurality of stack elements configured as latches and a second plurality of stack elements configured as a random access memory. The first and second plurality of stack elements and the central processing unit are provided in a single integrated circuit. A third plurality of stack elements is configured as a random access memory external to the single integrated circuit. In this aspect of the invention, desirably a first pointer is connected to the first plurality of stack elements, a second pointer connected to the second plurality of stack elements, and a third pointer is connected to the third plurality of stack elements. The central processing unit is connected to pop items from the first plurality of stack elements. The first stack pointer is connected to the second stack pointer to pop a first plurality of items from the second plurality of stack elements when the first plurality of stack elements are empty from successive pop operations by the central processing unit. The second stack pointer is connected to the third stack pointer to pop a second plurality of items from the third plurality of stack elements when the second plurality of stack elements are empty from successive pop operations by the central processing unit.

In another aspect of the invention, a first register is connected to supply a first input to the arithmetic logic unit. A first shifter is connected between an output of the arithmetic logic unit and the first register. A second register is connected to receive a starting polynomial value. An output of the second register is connected to a second shifter. A least significant bit of the second register is connected to the arithmetic logic unit. A third register is connected to supply feedback terms of a polynomial to the arithmetic logic unit. A down counter, for counting down a number corresponding to digits of a polynomial to be generated, is connected to the arithmetic logic unit. The arithmetic logic unit is responsive to a polynomial instruction to carry out an exclusive OR of the contents of the first register with the contents of the third register if the least significant bit of the second register is a "ONE"and to pass the contents of the first register unaltered if the least significant bit of the second register is a "ZERO", until the down counter completes a count. The polynomial to be generated results in said first register.

In still another aspect of the invention, a result register is connected to supply a first input to the arithmetic logic unit. A first, left shifting shifter is connected between an output of the arithmetic logic unit and the result register. A multiplier register is connected to receive a multiplier in bit reversed form. An output of the multiplier register is connected to a second, right shifting shifter. A least significant bit of the multiplier register is connected to the arithmetic logic unit. A third register is connected to supply a multiplicand to said arithmetic logic unit. A down counter, for counting down a number corresponding to one less than the number of digits of the multiplier, is connected to the arithmetic logic unit. The arithmetic logic unit is responsive to a multiply instruction to add the contents of the result register with the contents of the third register, when the least significant bit of the multiplier register is a "ONE" and to pass the contents of the result register unaltered, until the down counter completes a count. The product results in the result register.

The attainment of the foregoing and related objects, advantages and features of the invention should be more readily apparent to those skilled in the art, after review of the following more detailed description of the invention, taken together with the drawings, in which:

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is an external, plan view of an integrated circuit package incorporating a microprocessor in accordance with the invention.

FIG. 2 is a block diagram of a microprocessor in accordance with the invention.

FIG. 3 is a block diagram of a portion of a data processing system incorporating the microprocessor of FIGS. 1 and 2.

FIG. 4 is a more detailed block diagram of a portion of the microprocessor shown in FIG. 2.

FIG. 5 is a more detailed block diagram of another portion of the microprocessor shown in FIG. 2.

FIG. 6 is a block diagram of another portion of the data processing system shown in part in FIG. 3 and incorporating the microprocessor of FIGS. 1–2 and 4–5.

FIGS. 7 and 8 are layout diagrams for the data processing system shown in part in FIGS. 3 and 6.

FIG. 9 is a layout diagram of a second embodiment of a microprocessor in accordance with the invention in a data processing system on a single integrated circuit.

FIG. 10 is a more detailed block diagram of a portion of the data processing system of FIGS. 7 and 8.

FIG. 11 is a timing diagram useful for understanding operation of the system portion shown in FIG. 12.

FIG. 12 is another more detailed block diagram of a further portion of the data processing system of FIGS. 7 and 8.

FIG. 13 is a more detailed block diagram of a portion of the microprocessor shown in FIG. 2.

FIG. 14 is a more detailed block and schematic diagram of a portion of the system shown in FIGS. 3 and 7-8.

FIG. 15 is a graph useful for understanding operation of the system portion shown in FIG. 14.

FIG. 16 is a more detailed block diagram showing part of the system portion shown in FIG. 4.

FIG. 17 is a more detailed block diagram of a portion of the microprocessor shown in FIG. 2.

FIG. 18 is a more detailed block diagram of part of the microprocessor portion shown in FIG. 17.

FIG. 19 is a set of waveform diagrams useful for understanding operation of the part of the microprocessor portion shown in FIG. 18.

FIG. 20 is a more detailed block diagram showing another part of the system portion shown in FIG. 4.

FIG. 21 is a more detailed block diagram showing another part of the system portion shown in FIG. 4.

FIGS. 22 and 23 are more detailed block diagrams showing another part of the system portion shown in FIG. 4.

DETAILED DESCRIPTION OF THE INVENTION

Overview

The microprocessor of this invention is desirably implemented as a 32-bit microprocessor optimized for:

HIGH EXECUTION SPEED, and

LOW SYSTEM COST.

In this embodiment, the microprocessor can be thought of as 20 MIPS for 20 dollars. Important distinguishing features of the microprocessor are:

Uses low-cost commodity DYNAMIC RAMS to run 20 MIPS

4 instruction fetch per memory cycle

On-chip fast page-mode memory management

Runs fast without external cache

Requires few interfacing chips

Crams 32-bit CPU in 44 pin SOJ package

The instruction set is organized so that most operations can be specified with 8-bit instructions. Two positive products of this philosophy are:

Programs are smaller,

Programs can execute much faster.

The bottleneck in most computer systems is the memory bus. The bus is used to fetch instructions and fetch and store data. The ability to fetch four instructions in a single memory bus cycle significantly increases the bus availability to handle data.

Turning now to the drawings, more particularly to FIG. 1, there is shown a packaged 32-bit microprocessor **50** in a 44-pin plastic leadless chip carrier, shown approximately 100 times its actual size of about 0.8 inch on a side. The fact that the microprocessor **50** is provided as a 44-pin package represents a substantial departure from typical microprocessor packages, which usually have about 200 input/output (I/O) pins. The microprocessor **50** is rated at 20 million instructions per second (MIPS). Address and data lines **52**, also labelled D0–D31, are shared for addresses and data without speed penalty as a result of the manner in which the microprocessor **50** operates, as will be explained below.

Dynamic Ram

In addition to the low cost 44-pin package, another unusual aspect of the high performance microprocessor **50** is that it operates directly with dynamic random access memories (DRAMs), as shown by row address strobe (RAS) and column address strobe (CAS) I/O pins **54**. The other I/O pins for the microprocessor **50** include VDD pins **56**, VSS pins **58**, output enable pin **60**, write pin **62**, clock pin **64** and reset pin **66**.

All high speed computers require high speed and expensive memory to keep up. The highest speed static RAM memories cost as much as ten times as much as slower dynamic RAMs. This microprocessor has been optimized to use low-cost dynamic RAM in high-speed page-mode. Page-mode dynamic RAMs offer static RAM performance without the cost penalty. For example, low-cost 85 nsec. dynamic RAMs access at 25 nsec when operated in fast page-mode. Integrated fast page-mode control on the microprocessor chip simplifies system interfacing and results in a faster system.

Details of the microprocessor **50** are shown in FIG. 2. The microprocessor **50** includes a main central processing unit (CPU) **70** and a separate direct memory access (DMA) CPU **72** in a single integrated circuit making up the microprocessor **50**. The main CPU **70** has a first **16** deep push down stack **74**, which has a top item register **76** and a next item register **78**, respectively connected to provide inputs to an arithmetic logic unit (ALU) **80** by lines **82** and **84**. An output of the ALU **80** is connected to the top item register **76** by line **86**. The output of the top item register at **82** is also connected by line **88** to an internal data bus **90**.

A loop counter **92** is connected to a decrementer **94** by lines **96** and **98**. The loop counter **92** is bidirectionally connected to the internal data bus **90** by line **100**. Stack pointer **102**, return stack pointer **104**, mode register **106** and instruction register **108** are also connected to the internal data bus **90** by lines **110**, **112**, **114** and **116**, respectively. The internal data bus **90** is connected to memory controller **118** and to gate **120**. The gate **120** provides inputs on lines **122**, **124**, and **126** to X register **128**, program counter **130** and Y register **132** of return push down stack **134**. The X register **128**, program counter **130** and Y register **132** provide outputs to internal address bus **136** on lines **138**, **140** and **142**. The internal address bus provides inputs to the memory controller **118** and to an incrementer **144**. The incrementer **144** provides inputs to the X register, program counter and Y register via lines **146**, **122**, **124** and **126**. The DMA CPU **72** provides inputs to the memory controller **118** on line **148**. The memory controller **118** is connected to a RAM (not shown) by address/data bus **151** and control lines **153**.

FIG. 2 shows that the microprocessor **50** has a simple architecture. Prior art RISC microprocessors are substantially more complex in design. For example, the SPARC RISC microprocessor has three times the gates of the microprocessor **50**, and the Intel 8960 RISC microprocessor has 20 times the gates of the microprocessor **50**. The speed of this microprocessor is in substantial part due to this simplicity. The architecture incorporates push down stacks and register write to achieve this simplicity.

The microprocessor **50** incorporates an I/O that has been tuned to make heavy use of resources provided on

the integrated circuit chip. On chip latches allow use of the same I/O circuits to handle three different things: column addressing, row addressing and data, with a slight to non-existent speed penalty. This triple bus multiplexing results in fewer buffers to expand, fewer interconnection lines, fewer I/O pins and fewer internal buffers

The provision of on-chip DRAM control gives a performance equal to that obtained with the use of static RAMs. As a result, memory is provided at ¼ the system cost of static RAM used in most RISC systems.

The microprocessor **50** fetches 4 instructions per memory cycle; the instructions are in an 8-bit format, and this is a 32-bit microprocessor. System speed is therefore 4 times the memory bus bandwidth. This ability enables the microprocessor to break the Von Neumann bottleneck of the speed of getting the next instruction. This mode of operation is possible because of the use of a push down stack and register array. The push down stack allows the use of implied addresses, rather than the prior art technique of explicit addresses for two sources and a destination.

Most instructions execute in 20 nanoseconds in the microprocessor **50**. The microprocessor can therefore execute instructions at 50 peak MIPS without pipeline delays. This is a function of the small number of gates in the microprocessor **50** and the high degree of parallelism in the architecture of the microprocessor.

FIG. 3 shows how column and row addresses are multiplexed on lines D8–D14 of the microprocessor **50** for addressing DRAM 150 from I/O pins **52**. The DRAM 150 is one of eight, but only one DRAM 150 has been shown for clarity. As shown, the lines D11–D18 are respectively connected to row address inputs A0–A8 of the DRAM **150**. Additionally, lines D12–D15 are connected to the data inputs DQ1–DQ4 of the DRAM **15**. The output enable, write and column address strobe pins **54** are respectively connected to the output enable, write and column address strobe inputs of the DRAM **150** by lines **152**. The row address strobe pin **54** is connected through row address strobe decode logic **154** to the row address strobe input of the DRAM **150** by lines **156** and **158**.

D0–D7 pins **52** (FIG. 1) are idle when the microprocessor **50** is outputting multiplexed row and column addresses on D11–D18 pins **52**. The D0–D7 pins **52** can therefore simultaneously be used for I/O when right justified I/O is desired. Simultaneous addressing and I/O can therefore be carried out.

FIG. **4** shows how the microprocessor **50** is able to achieve performance equal to the use of static RAMS with DRAMs through multiple instruction fetch in a single clock cycle and instruction fetch-ahead. Instruction register **108** receives four 8-bit byte instruction words **1–4** on 32-bit internal data bus **90**. The four instruction byte **1–4** locations of the instruction register **108** are connected to multiplexer **170** by busses **172, 174, 176** and **178**, respectively. A microprogram counter **180** is connected to the multiplexer **170** by lines **182**. The multiplexer **170** is connected to decoder **184** by bus **186**. The decoder **184** provides internal signals to the rest of the microprocessor **50** on lines **188**.

Most significant bits **190** of each instruction byte **14** location are connected to a 4-input decoder **192** by lines **194**. The output of decoder **192** is connected to memory controller **118** by line **196**. Program counter **130** is connected to memory controller **118** by internal address bus **136**, and the instruction register **108** is connected to the memory controller **118** by the internal data bus **90**. Address/data bus **198** and control bus **200** are connected to the DRAMS **150** (FIG. 3).

In operation, when the most significant bits **190** of remaining instructions **1–4** are "1" in a clock cycle of the microprocessor **50**, there are no memory reference instructions in the queue. The output of decoder **192** on line **196** requests an instruction fetch ahead by memory controller **118** without interference with other accesses. While the current instructions in instruction register **108** are executing, the memory controller **118** obtains the address of the next set of four instructions from program counter **130** and obtains that set of instructions. By the time the current set of instructions has completed execution, the next set of instructions is ready for loading into the instruction register.

Details of the DMA CPU **72** are provided in FIG. 5. Internal data bus **90** is connected to memory controller **118** and to DMA instruction register **210**. The DMA instruction register **210** is connected to DMA program counter **212** by bus **214**, to transfer size counter **216** by bus **218** and to timed transfer interval counter **220** by bus **222**. The DMA instruction register **210** is also connected to DMA I/O and RAM address register **224** by line **226**. The DMA I/O and RAM address register **224** is connected to the memory controller **118** by memory cycle request line **228** and bus **230**. The DMA program counter **212** is connected to the internal address bus **136** by bus **232**. The transfer size counter **216** is connected to a DMA instruction done decrementer **234** by lines **236** and **238**. The decrementer **234** receives a control input on memory cycle acknowledge line **240**. When transfer size counter **216** has completed its count, it provides a control signal to DMA program counter **212** on line **242**. Timed transfer interval counter **220** is connected to decrementer **244** by lines **246** and **248**. The decrementer **244** receives a control input from a microprocessor system clock on line **250**.

The DMA CPU **72** controls itself and has the ability to fetch and execute instructions. It operates as a co-processor to the main CPU 70 (FIG. 2) for time specific processing.

FIG. **6** shows how the microprocessor **50** is connected to an electrically programmable read only memory (EPROM) **260** by reconfiguring the data lines **52** so that some of the data lines **52** are input lines and some of them are output lines. Data lines **52** D0–D7 provide data to and from corresponding data terminals **262** of the EPROM **260**. Data lines **52** D9–D18 provide addresses to address terminals **264** of the EPROM **260**. Data lines **52** D19–D31 provide inputs from the microprocessor **50** to memory and I/O decode logic **266**. RAS 0/1 control line **268** provides a control signal for determining whether the memory and I/O decode logic provides a DRAM RAS output on line **270** or a column enable output for the EPROM **260** on line **272**. Column address strobe terminal **60** of the microprocessor **50** provides an output enable signal on line **274** to the corresponding terminal **276** of the EPROM **260**.

FIGS. 7 and 8 show the front and back of a one card data processing system **280** incorporating the microprocessor **50**, MSM514258-10 type DRAMs **150** totalling 2 megabytes, a Motorola 50 MegaHertz crystal oscillator clock **282**, I/O circuits **284** and a 27256 type EPROM **260**. The I/O circuits **284** include a 74HC04 type high speed hex inverter circuit **286**, an IDT39C828 type 10-bit inverting buffer circuit **288**, an IDT39C822 type 10-bit inverting register circuit **290**, and two

IDT39C823 type 9-bit non-inverting register circuits **292**. The card **280** is completed with a MAX12V type DC-DC converter circuit **294**, 34-pin dual AMP type headers **296**, a coaxial female power connector **298**, and a 3-pin AMP right angle header **300**. The card **280** is a low cost, imbeddable product that can be incorporated in larger systems or used as an internal development tool.

The microprocessor **50** is a very high performance (50 MHz) RISC influenced 32-bit CPU designed to work closely with dynamic RAM. Clock for clock, the microprocessor **50** approaches the theoretical performance limits possible with a single CPU configuration. Eventually, the microprocessor **50** and any other processor is limited by the bus bandwidth and the number of bus paths. The critical conduit is between the CPU and memory.

One solution to the bus bandwidth/bus path problem is to integrate a CPU directly onto the memory chips, giving every memory a direct bus to the CPU. FIG. **9** shows another microprocessor **310** that is provided integrally with 1 megabit of DRAM **311** in a single integrated circuit **312**. Until the present invention, this solution has not been practical, because most high performance CPUs require from 500,000 to 1,000,000 transistors and enormous die sizes just by themselves. The microprocessor **310** is equivalent to the microprocessor **50** in FIGS. **1–8**. The microprocessors **50** and **310** are the most transistor efficient high performance CPUs in existence, requiring fewer than 50,000 transistors for dual processors **70** and **72** (FIG. **2**) or **314** and **316** (less memory). The very high speed of the microprocessors **50** and **310** is to a certain extent a function of the small number of active devices. In essence, the less silicon gets in the way, the faster the electrons can get where they are going.

The microprocessor **310** is therefore the only CPU suitable for integration on the memory chip die **312**. Some simple modifications to the basic microprocessor **50** to take advantage of the proximity to the DRAM array **311** can also increase the microprocessor **50** clock speed by 50 percent, and probably more.

The microprocessor **310** core on board the DRAM die **312** provides most of the speed and functionality required for a large group of applications from automotive to peripheral control. However, the integrated CPU **310**/DRAM **311** concept has the potential to redefine significantly the way multiprocessor solutions can solve a spectrum of very compute intensive problems. The CPU **310**/DRAM **311** combination eliminates the Von Neumann bottleneck by distributing it across numerous CPU/DRAM chips **312**. The microprocessor **310** is a particularly good core for multiprocessing, since it was designed with the SDI targeting array in mind, and provisions were made for efficient interprocessor communications.

Traditional multiprocessor implementations have been very expensive in addition to being unable to exploit fully the available CPU horsepower. Multiprocessor systems have typically been built up from numerous board level or box level computers. The result is usually an immense amount of hardware with corresponding wiring, power consumption and communications problems. By the time the systems are interconnected, as much as 50 percent of the bus speed has been utilized just getting through the interfaces.

In addition, multiprocessor system software has been scarce. A multiprocessor system can easily be crippled by an inadequate load-sharing algorithm in the system software, which allows one CPU to do a great deal of work and the others to be idle. Great strides have been made recently in systems software, and even UNIX V **4** may be enhanced to support multiprocessing. Several commercial products from such manufacturers as DUAL Systems and UNISOFT do a credible job on 68030 type microprocessor systems now.

The microprocessor **310** architecture eliminates most of the interface friction, since up to 64 CPU **310**/RAM **311** processors should be able to intercommunicate without buffers or latches. Each chip **312** has about 40 MIPS raw speed, because placing the DRAM **311** next to the CPU **310** allows the microprocessor **310** instruction cycle to be cut in half, compared to the microprocessor **50**. A 64 chip array of these chips **312** is more powerful than any other existing computer. Such an array fits on a 3×5 card, cost less than a FAX machine, and draw about the same power as a small television.

Dramatic changes in price/performance always reshape existing applications and almost always create new ones. The introduction of microprocessors in the mid 1970s created video games, personal computers, automotive computers, electronically controlled appliances, and low cost computer peripherals.

The integrated circuit **312** will find applications in all of the above areas, plus create some new ones. A common generic parallel processing algorithm handles convolution/Fast Fourier Transform (FFT)/pattern recognition. Interesting product possibilities using the integrated circuit **312** include high speed reading machines, real-time speech recognition, spoken language translation, real-time robot vision, a product to identify people by their faces, and an automotive or aviation collision avoidance system.

A real time processor for enhancing high density television (HDTV) images, or compressing the HDTV information into a smaller bandwidth, would be very feasible. The load sharing in HDTV could be very straightforward. Splitting up the task according to color and frame would require 6, 9 or 12 processors. Practical implementation might require 4 meg RAMs integrated with the microprocessor **310**.

The microprocessor **310** has the following specifications:

Control Lines
4—POWER/GROUND
1—CLOCK
32—DATA I/O
4—SYSTEM CONTROL
EXTERNAL MEMORY FETCH
EXTERNAL MEMORY FETCH AUTOINCREMENT X
EXTERNAL MEMORY FETCH AUTOINCREMENT Y
EXTERNAL MEMORY WRITE
EXTERNAL MEMORY WRITE AUTOINCREMENT X
EXTERNAL MEMORY WRITE AUTOINCREMENT Y
EXTERNAL PROM FETCH
LOAD ALL X REGISTERS
LOAD ALL Y REGISTERS
LOAD ALL PC REGISTERS
EXCHANGE X AND Y
INSTRUCTION FETCH
ADD TO PC
ADD TO X

WRITE MAPPING REGISTER
READ MAPPING REGISTER
REGISTER CONFIGURATION
MICROPROCESSOR **310** CPU **316** CORE
COLUMN LATCH1 (1024 BITS) 32×32 MUX
STACK POINTER (16 BITS)
COLUMN LATCH2 (1024 BITS) 32×32 MUX
RSTACK POINTER (16 BITS)
PROGRAM COUNTER 32 BITS
XO REGISTER 32 BITS (ACTIVATED ONLY FOR ON-CHIP ACCESSES)
YO REGISTER 32 BITS (ACTIVATED ONLY FOR ON-CHIP ACCESSES)
LOOP COUNTER 32 BITS
DMA CPU **314** CORE
DMA PROGRAM COUNTER 24 BITS
INSTRUCTION REGISTER 32 BITS
I/O & RAM ADDRESS REGISTER 32 BITS
Transfer Size Counter 12 Bits
Interval Counter 12 Bits

To offer memory expansion for the basic chip **312**, an intelligent DRAM can be produced. This chip will be optimized for high speed operation with the integrated circuit **312** by having three on-chip address registers: Program Counter, X Register and Y register. As a result, to access the intelligent DRAM, no address is required, and a total access cycle could be as short as 10 nsec. Each expansion DRAM would maintain its own copy of the three registers and would be identified by a code specifying its memory address. Incrementing and adding to the three registers will actually take place on the memory chips. A maximum of 64 intelligent DRAM peripherals would allow a large system to be created without sacrificing speed by introducing multiplexers or buffers.

There are certain differences between the microprocessor **310** and the microprocessor **50** that arise from providing the microprocessor **310** on the same die **312** with the DRAM **311**. Integrating the DRAM **311** allows architectural changes in the microprocessor **310** logic to take advantage of existing on-chip DRAM **311** circuitry. Row and column design is inherent in memory architecture. The DRAMs **311** access random bits in a memory array by first selecting a row of 1024 bits, storing them into a column latch, and then selecting one of the bits as the data to be read or written.

The time required to access the data is split between the row access and the column access. Selecting data already stored in a column latch is faster than selecting a random bit by at least a factor of six. The microprocessor **310** takes advantage of this high speed by creating a number of column latches and using them as caches and shift registers. Selecting a new row of information may be thought of as performing a 1024-bit read or write with the resulting immense bus bandwidth.

1. The microprocessor **50** treats its 32-bit instruction register **108** (see FIGS. **2** and **4**) as a cache for four 8-bit instructions. Since the DRAM **311** maintains a 1024-bit latch for the column bits, the microprocessor **310** treats the column latch as a cache for 128 8-bit instructions. Therefore, the next instruction will almost always be already present in the cache. Long loops within the cache are also possible and more useful than the 4 instruction loops in the microprocessor **50**.

2. The microprocessor **50** uses two 16×32-bit deep register arrays **74** and **134** (FIG. **2**) for the parameter stack and the return stack. The microprocessor **310** creates two other 1024-bit column latches to provide the equivalent of two 32×32-bit arrays, which can be accessed twice as fast as a register array.

3. The microprocessor **50** has a DMA capability which can be used for I/O to a video shift register. The microprocessor **310** uses yet another 1024-bit column latch as a long video shift register to drive a CRT display directly. For color displays, three on-chip shift registers could also be used. These shift registers can transfer pixels at a maximum of 100 MHz.

4. The microprocessor **50** accesses memory via an external 32-bit bus. Most of the memory **311** for the microprocessor **310** is on the same die **312**. External access to more memory is made using an 8-bit bus. The result is a smaller die, smaller package and lower power consumption than the microprocessor **50**.

5. The microprocessor **50** consumes about a third of its operating power charging and discharging the I/O pins and associated capacitances. The DRAMs **150** (FIG. **8**) connected to the microprocessor **50** dissipate most of their power in the I/O drivers. A microprocessor **310** system will consume about one-tenth the power of a microprocessor **50** system, since having the DRAM **311** next to the processor **310** eliminates most of the external capacitances to be charged and discharged.

6. Multiprocessing means splitting a computing task between numerous processors in order to speed up the solution. The popularity of multiprocessing is limited by the expense of current individual processors as well as the limited interprocessor communications ability. The microprocessor **310** is an excellent multiprocessor candidate, since the chip **312** is a monolithic computer complete with memory, rendering it low-cost and physically compact.

The shift registers implemented with the microprocessor **310** to perform video output can also be configured as interprocessor communication links. The INMOS transputer attempted a similar strategy, but at much lower speed and without the performance benefits inherent in the microprocessor **310** column latch architecture. Serial I/O is a prerequisite for many multiprocessor topologies because of the many neighbor processors which communicate. A cube has 6 neighbors. Each neighbor communicates using these lines:

DATA IN
CLOCK IN
READY FOR DATA
DATA OUT
DATA READY?
CLOCK OUT

A special start up sequence is used to initialize the on-chip DRAM **311** in each of the processors.

The microprocessor **310** column latch architecture allows neighbor processors to deliver information directly to internal registers or even instruction caches of other chips **312**. This technique is not used with existing processors, because it only improves performance in a tightly coupled DRAM system.

7. The microprocessor **50** architecture offers two types of looping structures: LOOP-IF-DONE and MICRO-LOOP. The former takes an 8-bit to **24**-bit operand to describe the entry point to the loop address. The latter performs a loop entirely within the 4 instruction queue and the loop entry point is implied as the first instruction in the queue. Loops entirely within the queue run without external instruction fetches and execute up to three times as fast as the long loop construct. The microprocessor **310** retains both constructs with a few differences. The microprocessor **310** microloop

functions in the same fashion as the microprocessor **50** operation, except the queue is 1024-bits or 128 8-bit instructions long. The microprocessor **310** microloop can therefore contain jumps, branches, calls and immediate operations not possible in the 4 8-bit instruction microprocessor **50** queue.

Microloops in the microprocessor **50** can only perform simple block move and compare functions. The larger microprocessor **310** queue allows entire digital signal processing or floating point algorithms to loop at high speed in the queue

The microprocessor **50** offers four instructions to redirect execution:

CALL
BRANCH
BRANCH-IF-ZERO
LOOP-IF-NOT-DONE

These instructions take a variable length address operand 8, 16 or 24 bits long. The microprocessor **50** next address logic treats the three operands similarly by adding or subtracting them to the current program counter. For the microprocessor **310**, the 16 and 24-bit operands function in the same manner as the 16 and 24-bit operands in the microprocessor **50**. The 8-bit class operands are reserved to operate entirely within the instruction queue. Next address decisions can therefore be made quickly, because only 10 bits of addresses are affected, rather than **32**. There is no carry or borrow generated past the 10 bits.

8. The microprocessor **310** CPU **316** resides on an already crowded DRAM die **312**. To keep chip size as small as possible, the DMA processor **72** of the microprocessor **50** has been replaced with a more traditional DMA controller **314**. DMA is used with the microprocessor **310** to perform the following functions:

Video output to a CRT
Multiprocessor serial communications
8-bit parallel I/O

The DMA controller **314** can maintain both serial and parallel transfers simultaneously. The following DMA sources and destinations are supported by the microprocessor **310**:

DESCRIPTION	I/O	LINES
1. Video shift register	OUTPUT	1 to 3
2. Multiprocessor serial	BOTH	6 lines/channel
3. 8-bit parallel	BOTH	8 data, 4 control

The three sources use separate 1024-bit buffers and separate I/O pins. Therefore, all three may be active simultaneously without interference.

The microprocessor **310** can be implemented with either a single multiprocessor serial buffer or separate receive and sending buffers for each channel, allowing simultaneous bidirectional communications with six neighbors simultaneously.

FIGS. **10** and **11** provide details of the PROM DMA used in the microprocessor **50**. The microprocessor **50** executes faster than all but the fastest PROMs. PROMS are used in a microprocessor **50** system to store program segments and perhaps entire programs. The microprocessor **50** provides a feature on power-up to allow programs to be loaded from low-cost, slow speed PROMs into high speed DRAM for execution. The logic which performs this function is part of the DMA memory controller **118**. The operation is similar to DMA, but not identical, since four 8-bit bytes must be assembled on the microprocessor **50** chip, then written to the DRAM **150**.

The microprocessor **50** directly interfaces to DRAM **150** over a triple multiplexed data and address bus **350**, which carries RAS addresses, CAS addresses and data. The EPROM **260**, on the other hand, is read with non-multiplexed busses. The microprocessor **50** therefore has a special mode which unmultiplexes the data and address lines to read 8 bits of EPROM data. Four 8-bit bytes are read in this fashion. The multiplexed bus **350** is turned back on, and the data is written to the DRAM **150**.

When the microprocessor **50** detects a RESET condition, the processor stops the main CPU **70** and forces a mode **0** (PROM LOAD) instruction into the DMA CPU **72** instruction register. The DMA instruction directs the memory controller to read the EPROM **260** data at 8 times the normal access time for memory. Assuming a 50 MHz microprocessor **50**, this means an access time of 320 nsec. The instruction also indicates:

The selection address of the EPROM **260** to be loaded,

The number of 32-bit words to transfer,

The DRAM **150** address to transfer into.

The sequence of activities to transfer one 32-bit word from EPROM **260** to DRAM 150 are:

1. RAS goes low at **352**, latching the EPROM **260** select information from the high order address bits. The EPROM **260** is selected.
2. Twelve address bits (consisting of what is normally DRAM CAS addresses plus two byte select bits are placed on the bus **350** going to the EPROM **260** address pins. These signals will remain on the lines until the data from the EPROM **260** has been read into the microprocessor **50**. For the first byte, the byte select bits will be binary **00**.
3. CAS goes low at **354**, enabling the EPROM **260** data onto the lower 8 bits of the external address-/data bus **350**. NOTE: It is important to recognize that, during this part of the cycle, the lower 8 bits of the external data/address bus are functioning as inputs, but the rest of the bus is still acting as outputs
4. The microprocessor **50** latches these eight least significant bits internally and shifts them 8 bits left to shift them to the next significant byte position.
5. Steps **2**, **3** and **4** are repeated with byte address **01**.
6. Steps **2**, **3** and **4** are repeated with byte address **10**.
7. Steps **2**, **3** and **4** are repeated with byte address **11**.
8. CAS goes high at **356**, taking the EPROM **260** off the data bus.
9. RAS goes high at **358**, indicating the end of the EPROM **260** access.
10. RAS goes low at **360**, latching the DRAM select information from the high order address bits. At the same time, the RAS address bits are latched into the DRAM **150**. The DRAM **150** is selected.
11. CAS goes low at **362**, latching the DRAM **150** CAS addresses.
12. The microprocessor **50** places the previously latched EPROM **260** 32-bit data onto the external address/data bus **350**. W goes low at **364**, writing the 32 bits into the DRAM **150**.
13. W goes high at **366**. CAS goes high at **368**. The process continues with the next word.

FIG. **12** shows details of the microprocessor **50** memory controller **118**. In operation, bus requests stay present until they are serviced. CPU **70** requests are priorit-

ized at **370** in the order of: 1, Parameter Stack; 2, Return Stack; 3, Data Fetch; 4 Instruction Fetch. The resulting CPU request signal and a DMA request signal are supplied as bus requests to bus control **372**, which provides a bus grant signal at **374**. Internal address bus **136** and a DMA counter **376** provide inputs to a multiplexer **378**. Either a row address or a column address are provided as an output to multiplexed address bus **380** as an output from the multiplexer **378**. The multiplexed address bus **380** and the internal data bus **90** provide address and data inputs, respectively, to multiplexer **382**. Shift register **384** supplies row address strobe (RAS) 1 and 2 control signals to multiplexer **386** and column address strobe (CAS) 1 and 2 control signals to multiplexer **388** on lines **390** and **392**. The shift register **384** also supplies output enable (OE) and write (W) signals on lines **394** and **396** and a control signal on line **398** to multiplexer **382**. The shift register **384** receives a RUN signal on line **400** to generate a memory cycle and supplies a MEMORY READY signal on line **402** when an access is complete.

Stack/Register Architecture

Most microprocessors use on-chip registers for temporary storage of variables. The on-chip registers access data faster than off-chip RAM. A few microprocessors use an on-chip push down stack for temporary storage.

A stack has the advantage of faster operation compared to on-chip registers by avoiding the necessity to select source and destination registers. (A math or logic operation always uses the top two stack items as source and the top of stack as destination.) The stack's disadvantage is that it makes some operations clumsy. Some compiler activities in particular require on-chip registers for efficiency.

As shown in FIG. **13**, the microprocessor **50** provides both on-chip registers **134** and a stack **74** and reaps the benefits of both

Benefits

1. Stack math and logic is twice as fast as those available on an equivalent register only machine. Most programmers and optimizing compilers can take advantage of this feature.
2. Sixteen registers are available for on-chip storage of local variables which can transfer to the stack for computation. The accessing of variables is three to four times as fast as available on a strictly stack machine.

The combined stack 74/register 134 architecture has not been used previously due to inadequate understanding by computer designers of optimizing compilers and the mix of transfer versus math/logic instructions.

ADAPTIVE MEMORY CONTROLLER

A microprocessor must be designed to work with small or large memory configurations. As more memory loads are added to the data, address, and control lines, the switching speed of the signals slows down. The microprocessor **50** multiplexes the address/data bus three ways, so timing between the phases is critical. A traditional approach to the problem allocates a wide margin of time between bus phases so that systems will work with small or large numbers of memory chips connected. A speed compromise of as much as **50%** is required.

As shown in FIG. **14**, the microprocessor **50** uses a feedback technique to allow the processor to adjust memory bus timing to be fast with small loads and slower with large ones. The OUTPUT ENABLE (OE) line **152** from the microprocessor **50** is connected to all memories **150** on the circuit board. The loading on the output enable line **152** to the microprocessor **50** is directly related to the number of memories **150** connected. By monitoring how rapidly OE **152** goes high after a read, the microprocessor **50** is able to determine when the data hold time has been satisfied and place the next address on the bus.

The level of the OE line **152** is monitored by CMOS input buffer **410** which generates an internal READY signal on line **412** to the microprocessor's memory controller. Curves **414** and **416** of the FIG. **15** graph show the difference in rise time likely to be encountered from a lightly to heavily loaded memory system. When the OE line **152** has reached a predetermined level to generate the READY signal, driver **418** generates an OUTPUT ENABLE signal on OE line **152**.

Skip Within The Instruction Cache

The microprocessor **50** fetches four 8-bit instructions each memory cycle and stores them in a 32-bit instruction register **108**, as shown in FIG. **16**. A class of "test and skip" instructions can very rapidly execute a very fast jump operation within the four instruction cache.

Skip Conditions

Always
ACC non-zero
ACC negative
Carry flag equal logic one
Never
ACC equal zero
ACC positive
Carry flag equal logic zero

The SKIP instruction can be located in any of the four byte positions **420** in the 32-bit instruction register **108**. If the test is successful, SKIP will jump over the remaining one, two, or three 8-bit instructions in the instruct ion register **108** and cause the next four-instruction group to be loaded into the register **108**. As shown, the SKIP operation is implemented by resetting the 2-bit microinstruction counter **180** to zero on line **422** and simultaneously latching the next instruction group into the register **108**. Any instructions following the SKIP in the instruction register are overwritten by the new instructions and not executed.

The advantage of SKIP is that optimizing compilers and smart programmers can often use it in place of the longer conditional JUMP instruction. SKIP also makes possible microloops which exit when the loop counts down or when the SKIP jumps to the next instruction group. The result is very fast code.

Other machines (such as the PDP-8 and Data General NOVA) provide the ability to skip a single instruction. The microprocessor **50** provides the ability to skip up to three instructions.

Microloop In The Instruction Cache

The microprocessor **50** provides the MICROLOOP instruction to execute repetitively from one to three instructions residing in the instruction register **108**. The microloop instruction works in conjunction with the LOOP COUNTER **92** (FIG. **2**) connected to the internal data bus **90**. To execute a microloop, the program stores a count in LOOP COUNTER **92**. MICROLOOP may be placed in the first, second, third, or last byte **420** of the instruction register **108**. If placed in the first position, execution will just create a delay equal to the

number stored in LOOP COUNTER **92** times the machine cycle. If placed in the second, third, or last byte **420**, when the microloop instruction is executed, it will test the LOOP COUNT for zero. If zero, execution will continue with the next instruction. If not zero, the LOOP COUNTER **92** is decremented and the 2-bit microinstruction counter is cleared, causing the preceding instructions in the instruction register to be executed again

Microloop is useful for block move and search operations. By executing a block move completely out of the instruction register **108**, the speed of the move is doubled, since all memory cycles are used by the move rather than being shared with instruction fetching. Such a hardware implementation of microloops is much faster than conventional software implementation of a comparable function.

Optimal CPU Clock Scheme

The designer of a high speed microprocessor must produce a product which operate over wide temperature ranges, wide voltage swings, and wide variations in semiconductor processing. Temperature, voltage, and process all affect transistor propagation delays. Traditional CPU designs are done so that with the worse case of the three parameters, the circuit will function at the rated clock speed. The result are designs that must be clocked a factor of two slower than their maximum theoretical performance, so they will operate properly in worse case conditions.

The microprocessor **50** uses the technique shown in FIGS. **17–19** to generate the system clock and its required phases. Clock circuit **430** is the familiar "ring oscillator" used to test process performance. The clock is fabricated on the same silicon chip as the rest of the microprocessor **50**.

The ring oscillator frequency is determined by the parameters of temperature, voltage, and process. At room temperature, the frequency will be in the neighborhood of 100 MHZ. At **70** degrees Centigrade, the speed will be 50 MHZ. The ring oscillator **430** is useful as a system clock, with its stages **431** producing phase O-phase **3** outputs **433** shown in FIG. **19**, because its performance tracks the parameters which similarly affect all other transistors on the same silicon die. By deriving system timing from the ring oscillator **430**, CPU **70** will always execute at the maximum frequency possible, but never too fast. For example, if the processing of a particular die is not good resulting in slow transistors, the latches and gates on the microprocessor **50** will operate slower than normal. Since the microprocessor **50** ring oscillator clock **430** is made from the same transistors on the same die as the latches and gates, it too will operate slower (oscillating at a lower frequency), providing compensation which allows the rest of the chip's logic to operate properly.

Asynchronous/Synchronous CPU

Most microprocessors derive all system timing from a single clock. The disadvantage is that different parts of the system can slow all operations. The microprocessor **50** provides a dual-clock scheme as shown in FIG. **17**, with the CPU **70** operating asynchronously to I/O interface **432** forming part of memory controller **118** (FIG. **2**) and the I/O interface **432** operating synchronously with the external world of memory and I/O devices. The CPU **70** executes at the fastest speed possible using the adaptive ring counter clock **430**. Speed may vary by a factor of four depending upon temperature, voltage, and process. The external world must be synchronized to the microprocessor **50** for operations such as video display updating and disc drive reading and writing. This synchronization is performed by the I/O interface **432**, speed of which is controlled by a conventional crystal clock **434**. The interface **432** processes requests for memory accesses from the microprocessor **50** and acknowledges the presence of I/O data. The microprocessor **50** fetches up to four instructions in a single memory cycle and can perform much useful work before requiring another memory access. By decoupling the variable speed of the CPU **70** from the fixed speed of the I/O interface **432**, optimum performance can be achieved by each. Recoupling between the CPU **70** and the interface **432** is accomplished with handshake signals on lines **436**, with data/addresses passing on bus **90, 136**.

Asynchronous/Synchronous CPU Imbedded On A Dram Chip

System performance is enhanced even more when the DRAM **311** and CPU **314** (FIG. **9**) are located on the same die. The proximity of the transistors means that DRAM **311** and CPU **314** parameters will closely follow each other. At room temperature, not only would the CPU **314** execute at 100 MHZ, but the DRAM **311** would access fast enough to keep up. The synchronization performed by the I/O interface **432** would be for DMA and reading and writing I/O ports. In some systems (such as calculators) no I/O synchronization at all would be required, and the I/O clock would be tied to the ring counter clock.

Variable Width Operands

Many microprocessors provide variable width operands. The microprocessor **50** handles operands of **8, 16**, or **24** bits using the same op-code. FIG. **20** shows the 32-bit instruction register **108** and the 2-bit microinstruction register **180** which selects the 8-bit instruction. Two classes of microprocessor **50** instructions can be greater than 8-bits, JUMP class and IMMEDIATE. A JUMP or IMMEDIATE op-code is 8-bits, but the operand can be 8, 16, or 24 bits long. This magic is possible because operands must be right justified in the instruction register. This means that the least significant bit of the operand is always located in the least significant bit of the instruction register. The microinstruction counter **180** selects which 8-bit instruction to execute. If a JUMP or IMMEDIATE instruction is decoded, the state of the 2-bit microinstruction counter selects the required 8, 16, or 24 bit operand onto the address or data bus. The unselected 8-bit bytes are loaded with zeros by operation of decoder **440** and gates **442**. The advantage of this technique is the saving of a number of op-codes required to specify the different operand sizes in other microprocessors.

Triple Stack Cache

Computer performance is directly related to the system memory bandwidth. The faster the memories, the faster the computer. Fast memories are expensive, so techniques have been developed to move a small amount of high-speed memory around to the memory addresses where it is needed. A large amount of slow memory is constantly updated by the fast memory, giving the appearance of a large fast memory array. A common implementation of the technique is known as a high-speed memory cache. The cache may be thought

of as fast acting shock absorber smoothing out the bumps in memory access. When more memory is required than the shock can absorb, it bottoms out and slow speed memory is accessed. Most memory operations can be handled by the shock absorber itself.

The microprocessor **50** architecture has the ALU **80** (FIG. **2**) directly coupled to the top two stack locations **76** and **78**. The access time of the stack **74** therefore directly affects the execution speed of the processor. The microprocessor **50** stack architecture is particularly suitable to a triple cache technique, shown in FIG. **21** which offers the appearance of a large stack memory operating at the speed of on-chip latches **450**. Latches **450** are the fastest form of memory device built on the chip, delivering data in as little as 3 nsec. However latches **450** require large numbers of transistors to construct. On-chip RAM **452** requires fewer transistors than latches, but is slower by a factor of five (15 nsec access). Off-chip RAM **150** is the slowest storage of all. The microprocessor **50** organizes the stack memory hierarchy as three interconnected stacks **450**, **452** and **454**. The latch stack **450** is the fastest and most frequently used. The on-chip RAM stack **452** is next. The off-chip RAM stack **454** is slowest. The stack modulation determines the effective access time of the stack. If a group of stack operations never push or pull more than four consecutive items on the stack, operations will be entirely performed in the 3 nsec latch stack. When the four latches **456** are filled, the data in the bottom of the latch stack **450** is written to the top of the on-chip RAM stack **452**. When the sixteen locations **458** in the on-chip RAM stack **452** are filled, the data in the bottom of the on-chip RAM stack **452** is written to the top of the off-chip RAM stack **454**. When popping data off a full stack **450**, four pops will be performed before stack empty line **460** from the latch stack pointer **462** transfers data from the on-chip RAM stack **452**. By waiting for the latch stack **450** to empty before performing the slower on-chip RAM access, the high effective speed of the latches **456** are made available to the processor. The same approach is employed with the on-chip RAM stack **452** and the off-chip RAM stack **454**.

Polynomial Generation Instruction

Polynomials are useful for error correction, encryption, data compression, and fractal generation. A polynomial is generated by a sequence of shift and exclusive OR operations. Special chips are provided for this purpose in the prior art.

The microprocessor **50** is able to generate polynomials at high speed without external hardware by slightly modifying how the ALU **80** works. As shown in FIG. **22**, a polynomial is generated by loading the "order" (also known as the feedback terms) into C Register **470**. The value thirty one (resulting in 32 is loaded into DOWN COUNTER **472**. A register **474** is loaded with zero. B register **476** is loaded with the starting polynomial value. When the POLY instruction executes, C register **470** is exclusively ORed with A register **474** if the least significant bit of B register **476** is a one. Otherwise, the contents of the A register **474** passes through the ALU **80** unaltered. The combination of A and B is then shifted right (divided by **2**) with shifters **478** and **480**. The operation automatically repeats the specified number of iterations, and the resulting polynomial is left in A register **474**.

Fast Multiply

Most microprocessors offer a 16×16 or 32×32 bit multiply instruction. Multiply when performed sequentially takes one shift/add per bit, or 32 cycles for 32 bit data. The microprocessor **50** provides a high speed multiply which allows multiplication by small numbers using only a small number of cycles. FIG. **23** shows the logic used to implement the high speed algorithm. To perform a multiply, the size of the multiplier less one is placed in the DOWN COUNTER **472**. For a four bit multiplier, the number three would be stored in the DOWN COUNTER **472**. Zero is loaded into the A register **474**. The multiplier is written bit reversed into the B Register **476**. For example, a bit reversed five (binary **0101**) would be written into B as **1010**. The multiplicand is written into the C register **470**. Executing the FAST MULT instruction will leave the result in the A Register **474**, when the count has been completed. The fast multiply instruction is important because many applications scale one number by a much smaller number. The difference in speed between multiplying a 32×32 bit and a 32×4 bit is a factor of **8**. If the least significant bit of the multiplier is a "ONE" the contents of the A register **474** and the C register **470** are added. If the least significant bit of the multiplier is a "ZERO" the contents of the A register are passed through the ALU **80** unaltered. The output of the ALU **80** is shifted left by shifter **482** in each iteration. The contents of the B register **476** are shifted right by the shifter **480** in each iteration.

INSTRUCTION EXECUTION PHILOSOPHY

The microprocessor **50** uses high speed D latches in most of the speed critical areas. Slower on-chip RAM is used as secondary storage.

The microprocessor **50** philosophy of instruction **30** execution is to create a hierarchy of speed as follows:

Logic and D latch transfers	1 cycle	20 nsec
Math	2 cycles	40 nsec
Fetch/store on-chip RAM	2 cycles	40 nsec
Fetch store in current RAS page	4 cycles	80 nsec
Fetch/store with RAS cycle	11 cycles	220 nsec

With a 50 MHZ clock, many operations can be performed in 20 nsec. and almost everything else in 40 nsec.

To maximize speed, certain techniques in processor design have been used. They include:

Eliminating arithmetic operations on addresses,
Fetching up to four instructions per memory cycle,
Pipelineless instruction decoding,
Generating results before they are needed,
Use of three level stack caching.

PIPELINE PHILOSOPHY

Computer instructions are usually broken down into sequential pieces, for example: fetch, decode, register read, execute, and store. Each piece will require a single machine cycle. In most Reduced Instruction Set Computer (RISC) chips, instruction require from three to six cycles.

RISC instructions are very parallel. For example, each of 70 different instructions in the SPARC (SUN Computer's RISC chip) has five cycles. Using a technique called "pipelining", the different phases of consecutive instructions can be overlapped.

To understand pipelining, think of building five residential homes. Each home will require in sequence, a foundation, framing, plumbing and wiring, roofing, and interior finish. Assume that each activity takes one week. To build one house will take five weeks.

But what if you want to build an entire subdivision? You have only one of each work crew, but when the foundation men finish on the first house, you immediately start them on the second one, and so on. At the end of five weeks, the first home is complete, but you also have five foundations. If you have kept the framing, plumbing, roofing, and interior guys all busy, from five weeks on, a new house will be completed each week.

This is the way a RISC chip like SPARC appears to execute an instruction in a single machine cycle. In reality, a RISC chip is executing one fifth of five instructions each machine cycle. And if five instructions stay in sequence, an instruction will be completed each machine cycle.

The problems with a pipeline are keeping the pipe full with instructions. Each time an out of sequence instruction such as a BRANCH or CALL occurs, the pipe must be refilled with the next sequence. The resulting dead time to refill the pipeline can become substantial when many IF/THEN/ELSE statements or subroutines are encountered.

The Pipeline Approach

The microprocessor **50** has no pipeline as such. The approach of this microprocessor to speed is to overlap instruction fetching with execution of the previously fetched instruction(s). Beyond that, over half the instructions (the most common ones) execute entirely in a single machine cycle of 20 nsec. This is possible because:

1. Instruction decoding resolves in 2.5 nsec.

2. Incremented/decremented and some math values are calculated before they are needed, requiring only a latching signal to execute.

3. Slower memory is hidden from high speed operations by high-speed D latches which access in 4 nsec.

The disadvantage for this microprocessor is a more complex chip design process. The advantage for the chip user is faster ultimate throughput since pipeline stalls cannot exist. Pipeline synchronization with availability flag bits and other such pipeline handling is not required by this microprocessor.

For example, in some RISC machines an instruction which tests a status flag may have to wait for up to four cycles for the flag set by the previous instruction to be available to be tested. Hardware and software debugging is also somewhat easier because the user doesn't have to visualize five instructions simultaneously in the pipe.

OVERLAPPING INSTRUCTION FETCH/EXECUTE

The slowest procedure the microprocessor **50** performs is to access memory. Memory is accessed when data is read or written. Memory is also read when instructions are fetched. The microprocessor **50** is able to hide fetch of the next instruction behind the execution of the previously fetched instruction(s). The microprocessor **50** fetches instructions in 4-byte instruction groups. An instruction group may contain from one to four instructions. The amount of time required to execute the instruction group ranges from 4 cycles for simple instructions to 64 cycles for a multiply.

When a new instruction group is fetched, the microprocessor instruction decoder looks at the most significant bit of all four of the bytes. The most significant bit of an instruction determines if a memory access is required. For example, CALL, FETCH, and STORE all require a memory access to execute. If all four bytes have nonzero most significant bits, the microprocessor initiates the memory fetch of the next sequential 4-byte instruction group. When the last instruction in the group finishes executing, the next 4-byte instruction group is ready and waiting on the data bus needing only to be latched into the instruction register. If the 4-byte instruction group required four or more cycles to execute and the next sequential access was a column address strobe (CAS) cycle, the instruction fetch was completely overlapped with execution.

INTERNAL ARCHITECTURE

The microprocessor **50** architecture consists of the following:

PARAMETER STACK	<--> ALU* <-->	Y REGISTER RETURN STACK
<---32 BITS---> 16 DEEP Used for math and logic		<---32 BITS---> 16 DEEP Used for subroutine and interrupt return addresses as well as local variables
Push down stack Can overflow into off-chip RAM.		Push down stack Can overflow into off-chip RAM Can also be accessed relative to top of stack.
LOOP COUNTER	(32-bits, can decrement by 1) Used by class of test and loop instructions.	
X REGISTER	(32-bits, can increment or decrement by 4). Used to point to RAM locations.	
PROGRAM COUNTER	(32-bits increments by 4). Points to 4-byte instruction groups in RAM	
INSTRUCTION REG	(32-Bits). Holds 4-byte instruction groups while they are being decoded	

-continued

and executed.

*Math and logic operations use the TOP item and NEXT to top Parameter Stack items as the operands The result is pushed onto the Parameter Stack.
*Return addresses from subroutines are placed on the Return Stack The Y REGISTER is used as a pointer to RAM locations Since the Y REGISTER is the top item of the Return Stack nesting of indices is straightforward

MODE—A register with mode and status bits
Mode-Bits
Slow down memory accesses by **8** if "**1**". Run full speed if "0" (Provided for access to slow EPROM.)
Divide the system clock by **1023** if "**1**" to reduce power consumption. Run full speed if "**0**" (On-chip counters slow down if this bit is set.)
Enable external interrupt **1**
Enable external interrupt **2**
Enable external interrupt **3**
Enable external interrupt **4**.
Enable external interrupt **5**.
Enable external interrupt **6**.
Enable external interrupt 7.
On-Chip Memory Locations
MODE-BITS
DMA-POINTER
DMA-COUNTER
STACK-POINTER—Pointer into Parameter Stack
STACK-DEPTH—Depth of on-chip Parameter Stack
RSTACK-POINTER—Pointer into Return Stack
RSTACK-DEPTH—Depth of on-chip Return Stack

Addressing Mode High Points

The data bus is 32-bits wide. All memory fetches and stores are 32-bits. Memory bus addresses are 30 bits. The least significant 2 bits are used to select one-of-four bytes in some addressing modes. The Program Counter, X Register, and Y Register are implemented as D latches with their outputs going to the memory address bus and the bus incrementer/decrementer. Incrementing one of these registers can happen quickly, because the incremented value has already rippled through the inc/dec logic and need only be clocked into the latch. Branches and Calls are made to 32-bit word boundaries.

INSTRUCTION SET

32-Bit Instruction Format

The thirty two bit instructions are CALL, BRANCH, BRANCH-IF-ZERO, and LOOP-IF-NOT-DONE. These instructions require the calculation of an effective address. In many computers, the effective address is calculated by adding or subtracting an operand with the current Program Counter. This math operation requires from four to seven machine cycles to perform and can definitely bog down machine execution. The microprocessor's strategy is to perform the required math operation at assembly or linking time and do a much simpler "Increment to next page" or "Decrement to previous page" operation at run time. As a result, the microprocessor branches execute in a single cycle.

24-Bit Operand Form

With a 24-bit operand, the current page is considered to be defined by the most significant 6 bits of the Program Counter.

16-Bit Operand Form

QQQQQQQQ—Wwwwww
XX—YYYYYYYY—YYYYYYYY With a 16-bit operand, the current page is considered to be defined by the most significant 14 bits of the Program Counter.

8-Bit Operand Form

QQQQQQQQ—QQQQQQQQ—WwWwWw
XX—YYYYYYYY With an 8-bit operand, the current page is considered to be defined by the most significant 22 bits of the Program Counter.

QQQQQQQQ—Any 8-bit instruction
WWWWWW—Instruction op-code.
XX—Select how the address bits will be used:
00 - Make all high-order bits zero (Page zero addressing)
01 - Increment the high-order bits. (Use next page)
10 - Decrement the high-order bits. (Use previous page)
11 - Leave the high-order bits unchanged (Use current page)
YYYYYYYY - The address operand field. This field is always shifted left two bits (to generate a word rather than byte address) and loaded into the Program Counter. The microprocessor instruction decoder figures out the width of the operand field by the location of the instruction op-code in the four bytes.

The compiler or assembler will normally use the shortest operand required to reach the desired address so that the leading bytes can be used to hold other instructions. The effective address is calculated by combining:
The current Program Counter,
The 8, 16, or 24 bit address operand in the instruction,
Using one of the four allowed addressing modes

EXAMPLES OF EFFECTIVE ADDRESS CALCULATION

Example 1

Byte 1	Byte 2	Byte 3	Byte 4
QQQQQQQQ	QQQQQQQQ	00000011	10011000

The "QQQQQQQQs" in Byte **1** and **2** indicate space in the 4-byte memory fetch which could be hold two other instructions to be executed prior to the CALL instruction. Byte **3** indicates a CALL instruction (six zeros) in the current page (indicated by the 11 bits). Byte **4** indicates that the hexadecimal number **98** will be forced into the Program Counter bits **2** through **10**. (Remember, a CALL or BRANCH always goes to a

Byte 1	Byte 2	Byte 3	Byte 4
WWWWWW XX -	YYYYYYYY -	YYYYYYYY -	YYYYYYYY

word boundary so the two least significant bits are always set to zero). The effect of this instruction would be to CALL a subroutine at WORD location HEX **98** in the current page. The most significant 22 bits of the Program Counter define the current page and will be unchanged.

Example 2

Byte 1	Byte 2	Byte 3	Byte 4
000001 01	00000001	00000000	00000000

If we assume that the Program Counter was HEX **0000 0156** which is binary:

00000000 00000000 00000001 01010110=OLD PROGRAM COUNTER

Byte **1** indicates a BRANCH instruction op code (**000001**) and "**01**" indicates select the next page. Byte **2,3,** and **4** are the address operand. These 24-bits will be shifted to the left two places to define a WORD address. HEX **0156** shifted left two places is HEX **0558**. Since this is a 24-bit operand instruction, the most significant 6 bits of the Program Counter define the current page. These six bits will be incremented to select the next page. Executing this instruction will cause the Program Counter to be loaded with HEX **0400 0558** which is binary:

00000100 00000000 00000101 01011000=NEW PROGRAM COUNTER

Instructions

Call-Long

0000 00XX—YYYYYYYY—YYYYYYYY—YYYYYYY

Load the Program Counter with the effective WORD address specified. Push the current PC contents onto the RETURN STACK.

Other Effects: CARRY or modes, no effect. May cause Return Stack to force an external memory cycle if on-chip Return Stack is full.

Branch

0000 01XX—YYYYYYYY—Yyyyyyyy—yyyyyyyy

Load the Program Counter with the effective WORD address specified.

Other Effects: NONE

Branch-If-Zone

0000 10XX—YYYYYYYY—yyyyyyyy—yyyyyyyy

Test the TOP value on the Parameter Stack. If the value is equal to zero, load the Program Counter with the effective WORD address specified. If the TOP value is not equal to zero, increment the Program Counter and fetch and execute the next instruction.

Other Effects: NONE

Loop-If-Not-Done

0000 11YY—(XXXX XXXX)—(XXXX XXXX)—(XXXX XXXX)

If the LOOP COUNTER is not zero, load the Program Counter with the effective WORD address specified. If the LOOP COUNTER is zero, decrement the LOOP COUNTER, increment the Program Counter and fetch and execute the next instruction.

Other Effects: NONE

8-Bit Instructions Philosophy

Most of the work in the microprocessor **50** is done by the 8-bit instructions. Eight bit instructions are possible with the microprocessor because of the extensive use of implied stack addressing. Many 32-bit architectures use 8-bits to specify the operation to perform but use an additional 24-bits to specify two sources and a destination.

For math and logic operations, the microprocessor **50** exploits the inherent advantage of a stack by designating the source operand(s) as the top stack item and the next stack item. The math or logic operation is performed, the operands are popped from the stack, and the result is pushed back on the stack. The result is a very efficient utilization of instruction bits as well as registers. A comparable situation exists between Hewlett Packard calculators (which use a stack) and Texas Instrument calculators which don't. The identical operation on an HP will require one half to one third the keystrokes of the TI.

The availability of 8-bit instructions also allows another architectural innovation, the fetching of four instructions in a single 32-bit memory cycle. The advantages of fetching multiple instructions are:

Increased execution speed even with slow memories,

Similar performance to the Harvard (separate data and instruction busses) without the expense,

Opportunities to optimize groups of instructions,

The capability to perform loops within this mini-cache.

The microloops inside the four instruction group are effective for searches and block moves.

Skip Instructions

The microprocessor **50** fetches instructions in 32-bit chunks called 4-byte instruction groups. These four bytes may contain four 8-bit instructions or some mix of 8-bit and 16 or 24-bit instructions. SKIP instructions in the microprocessor skip any remaining instructions in a 4-byte instruction group and cause a memory fetch to get the next 4-byte instruction group. Conditional SKIPs when combined with 3-byte BRANCHES will create conditional BRANCHES. SKIPs may also be used in situations when no use can be made of the remaining bytes in a 4-instruction group. A SKIP executes in a single cycle, whereas a group of three NOPs would take three cycles.

Skip-Always—skip any remaining instructions in this 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group.

Skip-If-Zero—If the TOP item of the Parameter Stack is zero, skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the TOP item is not zero, execute the next sequential instruction.

Skip-If-Positive—If the TOP item of the Parameter Stack has a the most significant bit (the sign bit) equal to "**0**", skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the TOP item is not "**0**" execute the next sequential instruction.

Skip-If-No-Carry—If the CARRY flag from a SHIFT or arithmetic operation is not equal to "**1**", skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to

fetch the next 4-byte instruction group. If the CARRY is equal to "1", execute the next sequential instruction.

Skip-Never Execute the next sequential (NOP) instruction. (Delay one machine cycle)

Skip-If-Not-Zero—If the TOP item on the Parameter Stack is not equal to "0" skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the TOP item is equal 0" execute the next sequential instruction

Skip-If-Negative—If the TOP item on the Parameter Stack has its most significant bit (sign bit) set to "1", skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the TOP item has its most significant bit set to "0" execute the next sequential instruction.

Skip-If-Carry—If the CARRY flag is set to "1" as a result of SHIFT or arithmetic operation, skip any remaining instructions in the 4-byte instruction group Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the CARRY flag is "0" execute the next sequential instruction.

Microloops

Microloops are a unique feature of the microprocessor architecture which allows controlled looping within a 4-byte instruction group. A microloop instruction tests the LOOP COUNTER for "0" and may perform an additional test. If the LOOP COUNTER is not "0" and the test is met, instruction execution continues with the first instruction in the 4-byte instruction group, and the LOOP COUNTER is decremented A microloop instruction will usually be the last byte in a 4-byte instruction group, but it can be any byte. If the LOOP COUNTER is '0" or the test is not met, instruction execution continues with the next instruction If the microloop is the last byte in the 4-byte instruction group, the most significant 30-bits of the Program Counter are incremented and the next 4-byte instruction group is fetched from memory On a termination of the loop on LOOP COUNTER equal to "0" the LOOP COUNTER will remain at "0" Microloops allow short iterative work such as moves and searches to be performed without slowing down to fetch instructions from memory

Example

Byte 1	Byte 2
FETCH-VIA-X-AUTOINCREMENT	STORE-VIA-Y-AUTOINCREMENT
Byte 3	Byte 4
ULOOP-UNTIL-DONE	QQQQQQQQ

This example will perform a block move. To initiate the transfer, X will be loaded with the starting address of the source. Y will be loaded with the starting address of the destination. The LOOP COUNTER will be loaded with the number of 32-bit words to move The microloop will FETCH and STORE and count down the LOOP COUNTER until it reaches zero. QQQQQQQQ indicates any instruction can follow

Microloop Instructions

ULOOP-UNTIL-DONE—If the LOOP COUNTER is not "0", continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER If the LOOP COUNTER is "0" continue execution with the next instruction.

ULOOP-IF-ZERO—If the LOOP COUNTER is not "0" and the TOP item on the Parameter Stack is "0", continue execution with the first instruction in the 4-byte instruction group Decrement the LOOP COUNTER If the LOOP COUNTER is "0" or the TOP item is "1", continue execution with the next instruction.

Uloop-if-positive—If the LOOP COUNTER is not "0" and the most significant bit (sign bit) is "0", continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the TOP item is "1", continue execution with the next instruction.

Uloop-if-not-carry-clear—If the LOOP COUNTER is not "0" and the floating point exponents found in TOP and NEXT are not aligned, continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the exponents are aligned, continue execution with the next instruction This instruction is specifically designed for combination with special SHIFT instructions to align two floating point numbers.

ULoop-Never—(DECREMENT-LOOP-COUNTER) Decrement the LOOP COUNTER. Continue execution with the next instruction.

ULoop-If-Not-Zero—If the LOOP COUNTER is not "0" and the TOP item of the Parameter Stack is "0", continue execution with the first instruction in the 4-byte instruction group Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the TOP item is "1", continue execution with the next instruction.

ULoop-If-Negative—If the LOOP COUNTER is not "0" and the most significant bit (sign bit) of the TOP item of the Parameter Stack is "1", continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the most significant bit of the Parameter Stack is "0" continue execution with the next instruction.

ULoop-If-Carry-Set—If the LOOP COUNTER is not "0" and the exponents of the floating point numbers found in TOP and NEXT are not aligned, continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the exponents are aligned, continue execution with the next instruction.

Return From Subroutine Or Interrupt

Subroutine calls and interrupt acknowledgements cause a redirection of normal program execution. In both cases, the current Program Counter is pushed onto the Return Stack, so the microprocessor can return to its place in the program after executing the subroutine or interrupt service routine.

NOTE: When a CALL to subroutine or interrupt is acknowledged the Program Counter has already been incremented and is pointing to the 4-byte instruction group following the 4-byte group currently being executed. The instruction decoding logic allows the microprocessor to perform a test and execute a return conditional on the outcome of the test in a single cycle. A RETURN pops an address from the Return Stack and stores it to the Program Counter.

Return Instructions

Return-Always—Pop the top item from the Return Stack and transfer it to the Program Counter.

Return-If-Zero—If the TOP item on the Parameter Stack is "0" pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

return-If-Positive—If the most significant bit (sign bit) of the TOP item on the Parameter Stack is a "0" pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

Return-If-Carry-Clear—If the exponents of the floating point numbers found in TOP and NEXT are not aligned, pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

Return-Never—Execute the next instruction. (NOP)

Return-If-Not-Zero—If the TOP item on the Parameter Stack is not "0" pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

Return-If-Negative—If the most significant bit (sign bit) of the TOP item on the Parameter Stack is a "1" pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

Return-If-Carry-Set—If the exponents of the floating point numbers found in TOP and NEXT are aligned, pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

HANDLING MEMORY FROM DYNAMIC RAM

The microprocessor **50**, like any RISC type architecture, is optimized to handle as many operations as possible on-chip for maximum speed. External memory operations take from 80 nsec. to 220 nsec. compared with on-chip memory speeds of from 4 nsec. to 30 nsec. There are times when external memory must be accessed.

External memory is accessed using three registers:

X-Register—A 30-bit memory pointer which can be used for memory access and simultaneously incremented or decremented.

Y-Register—A 30-bit memory pointer which can be used for memory access and simultaneously incremented or decremented.

Program-Counter—A 30-bit memory pointer normally used to point to 4-byte instruction groups. External memory may be accessed at addresses relative to the PC. The operands are sometimes called "Immediate" or "Literal" in other computers. When used as memory pointer, the PC is also incremented after each operation.

Memory Load & Store Instructions

Fetch-Via-X—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. X is unchanged.

Fetch-Via-Y—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. Y is unchanged.

Fetch-Via-X-Autoincrement—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. After fetching, increment the most significant 30 bits of X to point to the next 32-bit word address.

Fetch-Via-Y-Autoincrement—Fetch the 32-bit memory content pointed to by Y and push it onto the Parameter Stack. After fetching, increment the most significant 30 bits of Y to point to the next 32-bit word address.

Fetch-Via-X-Autodecrement—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. After fetching, decrement the most significant 30 bits of X to point to the previous 32-bit word address.

Fetch-Via-Y-Autodecrement—Fetch the 32-bit memory content pointed to by Y and push it onto the Parameter Stack. After fetching, decrement the most significant 30 bits of Y to point to the previous 32-bit word address.

Store-Via-X—Pop the top item of the Parameter Stack and store it in the memory location pointed to by X. X is unchanged.

Store-Via-X-Autoincrement—Pop the top item of the Parameter Stack and store it in the memory location pointed to by Y. Y is unchanged. STORE-VIA-X-AUTOINCREMENT - Pop the top item of the Parameter Stack and store it in the memory location pointed to by X. After storing, increment the most significant 30 bits of X to point to the next 32-bit word address.

Store-Via-Y-Autoincrement—Pop the top item of the Parameter Stack and store it in the memory location pointed to by Y. After storing, increment the most significant 30 bits of Y to point to the next 32-bit word address.

Store-Via-X-Autodecrement—Pop the top item of the Parameter Stack and store it in the memory location pointed to by X. After storing, decrement the most significant **30** bits of X to point to the previous 32-bit word address.

Store-Via-Y-Autodecrement—Pop the top item of the Parameter Stack and store it in the memory location pointed to by Y. After storing, decrement the most significant 30 bits of Y to point to the previous 32-bit word address.

Fetch-Via-PC—Fetch the 32-bit memory content pointed to by the Program Counter and push it onto the Parameter Stack. After fetching, increment the most significant 30 bits of the Program Counter to point to the next 32-bit word address.

*NOTE When this instruction executes, the PC is pointing to the memory location following the

instruction. The effect is of loading a 32-bit immediate operand. This is an 8-bit instruction and therefore will be combined with other 8-bit instructions in a 4-byte instruction fetch. It is possible to have from one to four FETCH-VIA-PC instructions in a 4-byte instruction fetch. The PC increments after each execution of FETCH-VIA-PC, so it is possible to push four immediate operands on the stack. The four operands would be the found in the four memory locations following the instruction

Bye-Fetch-Via-X—Fetch the 32-bit memory content pointed to by the most significant 30 bits of X. Using the two least significant bits of X, select one of four bytes from the 32-bit memory fetch, right justify the byte in a 32-bit field and push the selected byte preceded by leading zeros onto the Parameter Stack

Byte-Store-Via-X—Fetch the 32-bit memory content pointed to by the most significant 30 bits of X. Pop the TOP item from the Parameter Stack. Using the two least significant bits of X place the least significant byte into the 32-bit memory data and write the 32-bit entity back to the location pointed to by the most significant 30 bits of X.

Other Effects Of Memory Access Instructions

Any FETCH instruction will push a value on the Parameter Stack **74**. If the on-chip stack is full, the stack will overflow into off-chip memory stack resulting in an additional memory cycle. Any STORE instruction will pop a value from the Parameter Stack **74**. If the on-chip stack is empty, a memory cycle will be generated to fetch a value from off-chip memory stack.

Handling On-Chip Variables

High-level languages often allow the creation of LOCAL VARIABLES. These variables are used by a particular procedure and discarded. In cases of nested procedures, layers of these variables must be maintained. On-chip storage is up to five times faster than off-chip RAM, so a means of keeping local variables on-chip can make operations run faster. The microprocessor **50** provides the capability for both on-chip storage of local variables and nesting of multiple levels of variables through the Return Stack.

The Return Stack **134** is implemented as **16** on-chip RAM locations. The most common use for the Return Stack **134** is storage of return addresses from subroutines and interrupt calls. The microprocessor allows these **16** locations to also be used as addressable registers. The **16** locations may be read and written by two instructions which indicate a Return Stack relative address from **0–15**. When high-level procedures are nested, the current procedure variables push the previous procedure variables further down the Return Stack **134**. Eventually, the Return Stack will automatically overflow into off-chip RAM.

On-Chip Variable Instructions

Read-Local-Variable XXXX—Read the XXXXth location relative to the top of the Return Stack. (XXXX is a binary number from **0000–1111**). Push the item read onto the Parameter Stack. OTHER EFFECTS: If the Parameter Stack is full, the push operation will cause a memory cycle to be generated as one item of the stack is automatically stored to external RAM. The logic which selects the location performs a modulo **16** subtraction. If four local variables have been pushed onto the Return Stack and an instruction attempts to READ the fifth item, unknown data will be returned

Write-Local-Variable XXXX—Pop the TOP item of the Parameter Stack and write it into the XXXXth location relative to the top of the Return Stack. (XXXX is a binary number from **0000–1111**.) OTHER EFFECTS: If the Parameter Stack is empty, the pop operation will cause a memory cycle to be generated to fetch the Parameter Stack item from external RAM. The logic which selects the location performs a modulo 16 subtraction. If four local variables have been pushed onto the Return Stack, and an instruction attempts to WRITE to the fifth item, it is possible to clobber return addresses or wreak other havoc.

Register and Flip-Flop Transfer And Push Instructions

Drop—Pop the TOP item from the Parameter Stack and discard it

Swap—Exchange the data in the TOP Parameter Stack location with the data in the NEXT Parameter Stack location

DUP—Duplicate the TOP item on the Parameter Stack and push it onto the Parameter Stack.

Push-Loop-CounterPush the value in LOOP COUNTER onto the Parameter Stack

Pop-RStack-Push-To-stack—Pop the top item from the Return Stack and push it onto the Parameter Stack.

Push-X-Reg—Push the value in the X Register onto the Parameter Stack.

Push-Stack-Pointer—Push the value of the Parameter Stack pointer onto the Parameter Stack.

Push-RStack-Pointer—Push the value of the Return Stack pointer onto the Return Stack.

Push-Mode-Bits—Push the value of the MODE REGISTER onto the Parameter Stack

Push-Input—Read the **10** dedicated input bits and push the value (right justified and padded with leading zeros) onto the Parameter Stack

Set-Loop-Counter—Pop the TOP value from the Parameter Stack and store it into LOOP COUNTER.

Pop-Stack-Push-To-RStack—Pop the TOP item from the Parameter Stack and push it onto the Return Stack.

Set-X-Reg—Pop the TOP item from the Parameter Stack and store it into the X Register.

Set-Stack-Pointer—Pop the TOP item from the Parameter Stack and store it into the Stack Pointer

Set-RStack-Pointer—Pop the TOP item from the Parameter Stack and store it into the Return Stack Pointer.

Set-Mode-Bits—Pop the TOP value from the Parameter Stack and store it into the MODE BITS.

Set-output—Pop the TOP item from the Parameter Stack and output it to the 10 dedicated output bits OTHER EFFECTS: Instructions which push or pop the Parameter Stack or Return Stack may cause a memory cycle as the stacks overflow back and forth between on-chip and off-chip memory

Loading A Short Literal

A special case of register transfer instruction is used to push an 8-bit literal onto the Parameter Stack. This instruction requires that the 8-bits to be pushed reside in the last byte of a 4-byte instruction group. The instruc-

tion op-code loading the literal may reside in ANY of the other three bytes in the instruction group.

Example

BYTE 1	BYTE 2	BYTE 3
LOAD-SHORT-LITERAL	QQQQQQQQ	QQQQQQQQ
BYTE 4		
00001111		

In this example, QQQQQQQQ indicates any other 8-bit instruction. When Byte 1 is executed, binary **00001111** (HEX 0f) from Byte 4 will be pushed (right justified and padded by leading zeros) onto the Parameter Stack. Then the instructions in Byte 2 and Byte 3 will execute. The microprocessor instruction decoder knows not to execute Byte 4. It is possible to push three identical 8-bit values as follows:

BYTE 1	BYTE 2
LOAD-SHORT-LITERAL	LOAD-SHORT-LITERAL
BYTE 3	BYTE 4
LOAD-SHORT-LITERAL	00001111

Short-literal-Instruction

Load-Short-Literal—Push the 8bit value found in Byte 4 of the current 4-byte instruction group onto the Parameter Stack

Logic Instructions

Logical and math operations used the stack for the source of one or two operands and as the destination for results. The stack organization is a particularly convenient arrangement for evaluating expressions. TOP indicates the top value on the Parameter Stack **74**. NEXT indicates the next to top value on the Parameter Stack **74**.

AND—Pop TOP and NEXT from the Parameter Stack, perform the logical AND operation on these two operands, and push the result onto the Parameter Stack.

OR—Pop TOP and NEXT from the Parameter Stack, perform the logical OR operation on these two operands, and push the result onto the Parameter Stack.

XOR—Pop TOP and NEXT from the Parameter Stack, perform the logical exclusive OR on these two operands, and push the result onto the Parameter Stack.

Bit-Clear—Pop TOP and NEXT from the Parameter Stack, toggle all bits in NEXT, perform the logical AND operation on TOP, and push the result onto the Parameter Stack. (Another way of understanding this instruction is thinking of it as clearing all bits in TOP that are set in NEXT.)

Math Instructions

Math instruction pop the TOP item and NEXT to top item of the Parameter Stack **74** to use as the operands. The results are pushed back on the Parameter Stack. The CARRY flag is used to latch the "33rd bit" of the ALU result.

Add—Pop the TOP item and NEXT to top item from the Parameter Stack, add the values together and push the result back on the Parameter Stack. The CARRY flag may be changed.

Add-With-Carry—Pop the TOP item and the NEXT to top item from the Parameter Stack, add the values together. If the CARRY flag is "1" increment the result. Push the ultimate result back on the Parameter Stack. The CARRY flag may be changed.

ADD-X—Pop the TOP item from the Parameter Stack and read the third item from the top of the Parameter Stack. Add the values together and push the result back on the Parameter Stack. The CARRY flag may be changed.

SUB—Pop the TOP item and NEXT to top item from the Parameter Stack, Subtract NEXT from TOP and push the result back on the Parameter Stack. The CARRY flag may be changed.

SUB-WITH-CARRY—Pop the TOP item and NEXT to top item from the Parameter Stack. Subtract NEXT from TOP. If the CARRY flag is "1" increment the result. Push the ultimate result back on the Parameter Stack. The CARRY flag may be changed.

SUB-X—

SIGNED-MULT-STEP—

UNSIGNED-MULT-STEP—

SIGNED-FAST-MULT—

FAST-MULT-STEP—

UNSIGNED-DIV-STEP—

GENERATE-POLYNOMIAL—

ROUND—

COMPARE—Pop the TOP item and NEXT to top item from the Parameter Stack. Subtract NEXT from TOP. If the result has the most significant bit equal to "0" (the result is positive), push the result onto the Parameter Stack. If the result has the most significant bit equal to "1" (the result is negative), push the old value of TOP onto the Parameter Stack. The CARRY flag may be affected.

Shift/Rotate

SHIFT-LEFT—Shift the TOP Parameter Stack item left one bit. The CARRY flag is shifted into the least significant bit of TOP.

SHIFT-RIGHT—Shift the TOP Parameter Stack item right one bit. The least significant bit of TOP is shifted into the CARRY flag. Zero is shifted into the most significant bit of TOP.

DOUBLE-SHIFT-LEFT—Treating the TOP item of the Parameter Stack as the most significant word of a 64-bit number and the NEXT stack item as the least significant word, shift the combined 64-bit entity left one bit. The CARRY flag is shifted into the least significant bit of NEXT.

DOUBLE-SHIFT-RIGHT—Treating the TOP item of the Parameter Stack as the most significant word of a 64-bit number and the NEXT stack item as the least significant word, shift the combined 64-bit entity right one bit. The least significant bit of NEXT is shifted into the CARRY flag. Zero is shifted into the most significant bit of TOP.

Other Instructions

FLUSH-STACK—Empty all on-chip Parameter Stack locations into off-chip RAM. (This instruction useful for multitasking applications). This instruction accesses a counter which holds the depth

of the on-chip stack and can require from none to 16 external memory cycles.

FLUSH-RSTACK—Empty all on-chip Return Stack locations into off-chip RAM. (This instruction is useful for multitasking applications). This instruction accesses a counter which holds the depth of the on-chip Return Stack and can require from none to **16** external memory cycles

It should further be apparent to those skilled in the art that various changes in form and details of the invention as shown and described may be made. It is intended that such changes be included within the spirit and scope of the claims appended hereto

What is claimed is:

1 A microprocessor system, comprising a central processing unit integrated circuit, a memory extend of said central processing unit integrated circuit, a bus connecting said central processing unit integrated circuit to said memory, and means connected to said bus for fetching instructions for said central processing unit integrated circuit on said bus from said memory, said means for fetching instructions being configured and connected to fetch multiple sequential instructions from said memory in parallel and supply the multiple sequential instructions to said central processing unit integrated circuit during a single memory cycle, said bus having a width at least equal to a number of bits in each of the instructions times a number of the instructions fetched in parallel, said central processing unit including an arithmetic logic unit and a first push down stack connected to said arithmetic, logic unit, said first push down stack including means for storing a top item connected to a first input of said arithmetic logic unit to provide the top item to the first input and means for storing a next item connected to a second input of said arithmetic logic unit to provide the next item to the second input, a remainder of said first push down stack being connected to said means for storing a next item to receive the next item from said means for storing a next item when pushed down in said push down stack said arithmetic logic unit having an output connected to said means for storing a top item.

2 The microprocessor system of claim **1** additionally comprising means connected to said means for fetching multiple instructions for determining by decoding the multiple instructions if multiple instructions fetched by said means for fetching multiple instructions require a memory access, said means for fetching multiple instructions fetching additional multiple instructions if decoding the multiple instructions shows that the multiple instructions do not require a memory access.

3 The microprocessor system of claim **2** in which the decoding determines if the multiple instructions do not require a memory access by a state of a bit of each of the multiple instructions. –

4. The microprocessor system of claim **3** in which the bit is a most significant bit of the multiple instructions.

5. The microprocessor system of claim **1** additionally comprising an instruction register for the multiple instructions connected to said means for fetching instructions, means connected to said instruction register for supplying the multiple instructions in succession from said instruction register, a counter connected to control said means for supplying the multiple instructions to supply the multiple instructions in succession, means for decoding the multiple instructions connected to receive the multiple instructions in succession from the means for supplying the multiple instructions, said counter being connected to said means for decoding to receive incrementing and reset control signals from said means for decoding, said means for decoding being configured to supply the reset control signal to said counter and to supply a control signal to said means for fetching instructions in response to a SKIP instruction in the multiple instructions

6 The microprocessor system of claim **5** additionally comprising a loop counter connected to receive a decrement control signal from said means for decoding, said means for decoding being configured to supply the reset control signal to said counter and the decrement control signal to said loop counter in response to a MICROLOOP instruction in the multiple instructions to provide a microloop within the multiple instructions in said instruction register for a number of repetitions controlled by said loop counter.

7. The microprocessor system of claim **1** additionally comprising an instruction register for the multiple instructions and a variable width operand to be used with one of the multiple instructions connected to said means for fetching instructions, means connected to said instruction register for supplying the multiple instructions in succession from said instruction register, a counter connected to control said means for supplying the multiple instructions to supply the multiple instructions in succession,

means for decoding the multiple instructions connected to receive the multiple instructions in succession from the means for supplying the multiple instructions, said counter being connected to said means for decoding to receive incrementing and reset control signals from said means for decoding, said means for decoding being configured to control said counter in response to an instruction utilizing the variable width operand stored in said instruction register, and means connected to said counter to select the variable width operand for use with the instruction utilizing the variable width operand in response to said counter.

8. A microprocessor system, comprising a central processing unit, a memory, a bus connecting said central processing unit to said memory, and means connected to said bus for fetching instructions for said central processing unit on said bus from said memory, said means for fetching instructions being configured and connected to fetch multiple sequential instructions from said memory in parallel and supply the multiple sequential instructions to said central processing unit during a single memory cycle. said central processing unit including an arithmetic logic unit and a first push down stack connected to said arithmetic logic unit, said first push down stack further including means for storing a top item connected to a first input of said arithmetic logic unit to provide the top item to the first input, means for storing a next item connected to a second input of said arithmetic logic unit to provide the next item to the second input, said arithmetic logic unit having an output connected to said means for storing a top item, a second push down stack, said means for storing a top item being connected to provide an input to said second push down stack and a control means connected between said means for storing a top item and said second push down stack for controlling provision of the input to said second push down stack, said second push down stack additionally being configured as a register file and said means for storing a top item and said sec-

ond push down stack additionally configured as the register file being bidirectionally connected

9 A microprocessor system, comprising a central processing unit, a dynamic random access memory, a bus connecting said central processing unit to said dynamic random access memory, and multiplexing means on said bus between said central processing unit and said dynamic random access memory, said multiplexing means being connected and configured to provide multiplexed row addresses, column addresses and data on said bus from said central processing unit to said dynamic random access memory and to provide data from said dynamic random access memory to said central processing unit, and

means connected to said bus for fetching instructions for said central processing unit on said bus from said dynamic random access memory, said means for fetching instructions being configured to fetch multiple sequential instructions from said dynamic random access memory in parallel and supply the multiple instructions to said central processing unit during a single memory cycle,

said central processing unit including an arithmetic logic unit and a first push down stack connected to said arithmetic logic unit, said first push down stack including means for storing a top item connected to a first input of said arithmetic logic unit to provide the top item to the first input, and means for storing a next item connected to a second input of said arithmetic logic unit to provide the next item to the second input, a remainder of said first push down stack being connected to said means for storing a next item to receive the next item from said means for storing a next item when pushed down in said push down stack, said arithmetic logic unit having an output connected to said means for storing a top item.

10. The microprocessor system of claim 9 additionally comprising a second push down stack, said means for storing a top item being connected to provide an input to said second push down stack and a control means connected between said means for storing a top item and said second push down stack for controlling provision of the input to said second push down stack.

11. The microprocessor system of claim 10 in which said second push down stack is additionally configured as a register file and said means for storing a top item and said second push down stack additionally configured as the register file are bidirectionally connected.

12. The microprocessor system of claim 11 additionally comprising means connected to said means for fetching multiple instructions for determining by decoding the multiple instructions if multiple instructions fetched by said means for fetching multiple instructions require a memory access, said means for fetching multiple instructions fetching additional multiple instructions if decoding the multiple instructions shows that the multiple instructions do not require a memory access.

13. The microprocessor system of claim 12 additionally comprising an instruction register for the multiple instructions connected to said means for fetching instructions, means connected to said instruction register for supplying the multiple instructions in succession from said instruction register, a counter connected to control said means for supplying the multiple instructions to supply the multiple instructions in succession, means for decoding the multiple instructions connected to receive the multiple instructions in succession from the means for supplying the multiple instructions, said counter being connected to said means for decoding to receive incrementing and reset control signals from said means for decoding, said means for decoding being configured to supply the reset control signal to said counter and to supply a control signal to said means for fetching instructions in response to a SKIP instruction in the multiple instructions

14. The microprocessor system of claim 13 additionally comprising a loop counter connected to receive a decrement control signal from said means for decoding, said means for decoding being configured to supply the reset control signal to said counter and the decrement control signal to said loop counter in response to a MICROLOOP instruction in the multiple instructions within the multiple instructions in said instruction register for a number of repetitions controlled by said loop counter.

15. The microprocessor system of claim 13 in which said means for decoding is configured to control said counter in response to one of the multiple instructions utilizing a variable width operand stored in said instruction register with the multiple instructions, said microprocessor system additionally comprising means connected to said counter to select the variable width operand for use with the instruction utilizing the variable width operand in response to a state of said counter resulting from control of said counter by said means for decoding.

16. The microprocessor system of claim 12 in which the decoding determines if the multiple instructions do not require a memory access by a state of a bit of each of the multiple instructions

17. The microprocessor system of claim 16 in which the bit is a most significant bit of the multiple instructions.

18. The microprocessor system of claim 9 additionally comprising a programmable read only memory containing instructions connected to said bus, means connected to said bus for fetching instructions for said central processing unit on said bus, said means for fetching instructions including means for assembling a plurality of instructions from said programmable read only memory, storing the plurality of instructions in said dynamic random access memory and subsequently supplying the plurality of instructions from said dynamic random access memory to said central processing unit on said bus

19 The microprocessor system of claim 9 additionally comprising a direct memory access processing unit having the capacity to request and execute instructions, said bus connecting said direct memory access processing unit to said dynamic random access memory, said dynamic random access memory containing instructions for said central processing unit and said direct memory access processing unit, said direct memory access processing unit being connected to means for fetching instructions for said central processing unit on said bus and for fetching instructions for said direct memory access processing unit on said bus.

20 The microprocessor system of claim 19 additionally comprising a variable speed system clock connected to said central processing unit and a fixed speed system clock connected to control said means for fetching instructions for said central processing unit and for fetching instructions for said direct memory access processing unit.

21. The microprocessor system of claim 9 in which said microprocessor system is configured to provide

different memory access timing for different storing capacity sizes of said dynamic random access memory by including a sensing circuit and a driver circuit, and an output enable line connected between said dynamic random access memory, said sensing circuit and said driver circuit, said sensing circuit being configured to provide a ready signal when said output enable line reaches a predetermined electrical level after a memory read operation as a function of different capacitance on said bus as a result of the different storing capacity sizes of said dynamic random access memory, said microprocessor system being configured so that said driver circuit provides an enabling signal on said output enable line responsive to the ready signal.

22. The microprocessor system of claim 21 in which the predetermined electrical level is a predetermined voltage.

23. The microprocessor system of claim 9 in which said microprocessor system is configured to operate at a variable clock speed; said microprocessor system additionally comprising a ring counter variable speed system clock connected to said central processing unit, said central processing unit and said ring counter variable speed system clock being provided in a single integrated circuit, said ring counter variable speed system clock being configured to provide different clock speed to said central processing unit as a result of transistor propagation delays, depending on at least one of temperature of said single integrated circuit, voltage and microprocessor fabrication process for said single integrated circuit.

24. The microprocessor system of claim 23 additionally comprising an input/output interface connected between said microprocessor system and an external memory bus to exchange coupling control signals, addresses and data between said central processing unit and said input/output interface, and a second clock independent of said ring counter variable speed system clock connected t said input/output interface to provide clock signals for operation of said input/output interface asynchronously from said central processing unit.

25. The microprocessor system of claim 24 in which said second clock is a fixed frequency clock.

26. The microprocessor system of claim 9 in which said first push down stack has a first plurality of stack registers having stack memory elements configured as latches, a second plurality of stack registers having stack memory elements configured as a random access memory, said first and second plurality of stack registers and said central processing unit being provided in a single integrated circuit with a top one of said second plurality of stack registers being connected to said a bottom one of said first plurality of stack registers, and a third plurality of stack registers having stack memory elements configured as a random access memory external to said single integrated circuit, with a top one of said third plurality of stack registers being connected to a bottom one of said second plurality of stack registers, said microprocessor system being configured to operate said first, second and third plurality of stack registers hierarchically as interconnected stacks.

27. The microprocessor system of claim 26 additionally comprising a first pointer connected to said first plurality of stack registers, a second pointer connected to said second plurality of stack registers, and a third pointer connected to said third plurality of stack registers, said microprocessor system being configured to operate said first, second and third plurality of stack registers hierarchically as interconnected stacks by having said central processing unit being connected to pop items from said first plurality of stack registers, said first stack pointer being connected to said second stack pointer to pop a first plurality of items from said second plurality of stack registers when said first plurality of stack registers are empty from successive pop operations by said central processing unit, said second stack pointer being connected to said third stack pointer to pop a second plurality of items from said third plurality of stack registers when said second plurality of stack registers are empty from successive pop operations by said central processing unit.

28. The microprocessor system of claim 9 additionally comprising a first register connected to supply a first input to said arithmetic logic unit, a first shifter connected between an output of said arithmetic logic unit and said first register, a second register connected to receive a starting polynomial value, an output of said second register being connected to a second shifter, a least significant bit of said second register being connected to said arithmetic logic unit, a third register connected to supply feedback terms of a polynomial to said arithmetic logic unit, a down counter, for counting down a number corresponding to digits of a polynomial to be generated, connected to said arithmetic logic unit, said arithmetic logic unit being responsive to a polynomial instruction to carry out an exclusive OR of the contents of said first register with the contents of said third register if the least significant bit of said second register is a "ONE" and to pass the contents of said first register unaltered if the least significant bit of said second register is a "ZERO" until said down counter completes a count, the polynomial to be generated resulting in said first register.

29. The microprocessor system of claim 28 in which said first register is a result register, said first shifter is a left shifting shifter, said second register is a multiplier register connected to receive a multiplier in bit reversed form, said second shifter is a right shifting shifter, said third register is connected to supply a multiplicand to said arithmetic logic unit, said down counter is configured for counting down a number corresponding to one less than the number of digits of the multiplier, said arithmetic logic unit being responsive to a multiply instruction to add the contents of said result register with the contents of said third register, if the least significant bit of said second register is a "ONE" and to pass the contents of said first register unaltered if the least significant bit of said second register is a "ZERO" until said down counter completes a count, the product resulting in said first register.

* * * * *

US006598148B1

(12) **United States Patent**
Moore et al.

(10) **Patent No.: US 6,598,148 B1**
(45) **Date of Patent: Jul. 22, 2003**

(54) **HIGH PERFORMANCE MICROPROCESSOR HAVING VARIABLE SPEED SYSTEM CLOCK**

(75) Inventors: **Charles H. Moore**, Woodside, CA (US); **Russell H. Fish, III**, Dallas, TX (US)

(73) Assignee: **Patriot Scientific Corporation** Poway, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days

(21) Appl. No.: **09/124,623**

(22) Filed: **Jul. 29, 1998**

(Under 37 CFR 1.47)

Related U.S. Application Data

(62) Division of application No. 08/484,918, filed on Jun. 7, 1995, now Pat. No. 5,809,336 which is a division of application No. 07/389,334 filed on Aug. 3, 1989 now Pat. No. 5,440,749

(51) **Int. Cl.**[7] **G06F 15/00**

(52) **U.S. Cl.** **712/32**

(58) **Field of Search** 712/32; 711/104, 711/105

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,680,698	A *	7/1987	Edwards et al.	712/37
5,379,438	A *	1/1995	Bell et al.	712/37

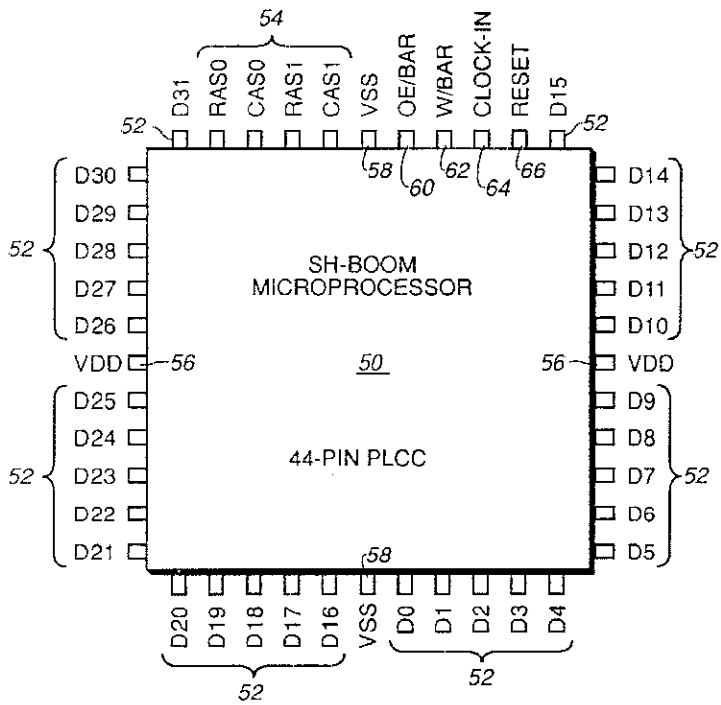
* cited by examiner

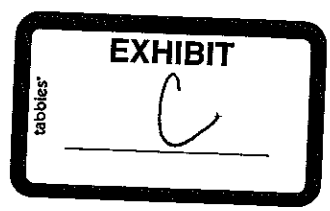
Primary Examiner—David Y. Eng
(74) *Attorney, Agent, or Firm*—Knobbe Martens Olson & Bear LLP

(57) **ABSTRACT**

A microprocessor integrated circuit including a processing unit disposed upon an integrated circuit substrate is disclosed herein. The processing unit is designed to operate in accordance with a predefined sequence of program instructions stored within an instruction register. A memory, capable of storing information provided by the processing unit and occupying a larger area of the integrated circuit substrate than the processing unit, is also provided within the microprocessor integrated circuit. The memory may be implemented using, for example dynamic or static random-access memory. A variable output frequency system clock, such as generated by a ring oscillator, is also disposed on the integrated circuit substrate.

13 Claims, 19 Drawing Sheets





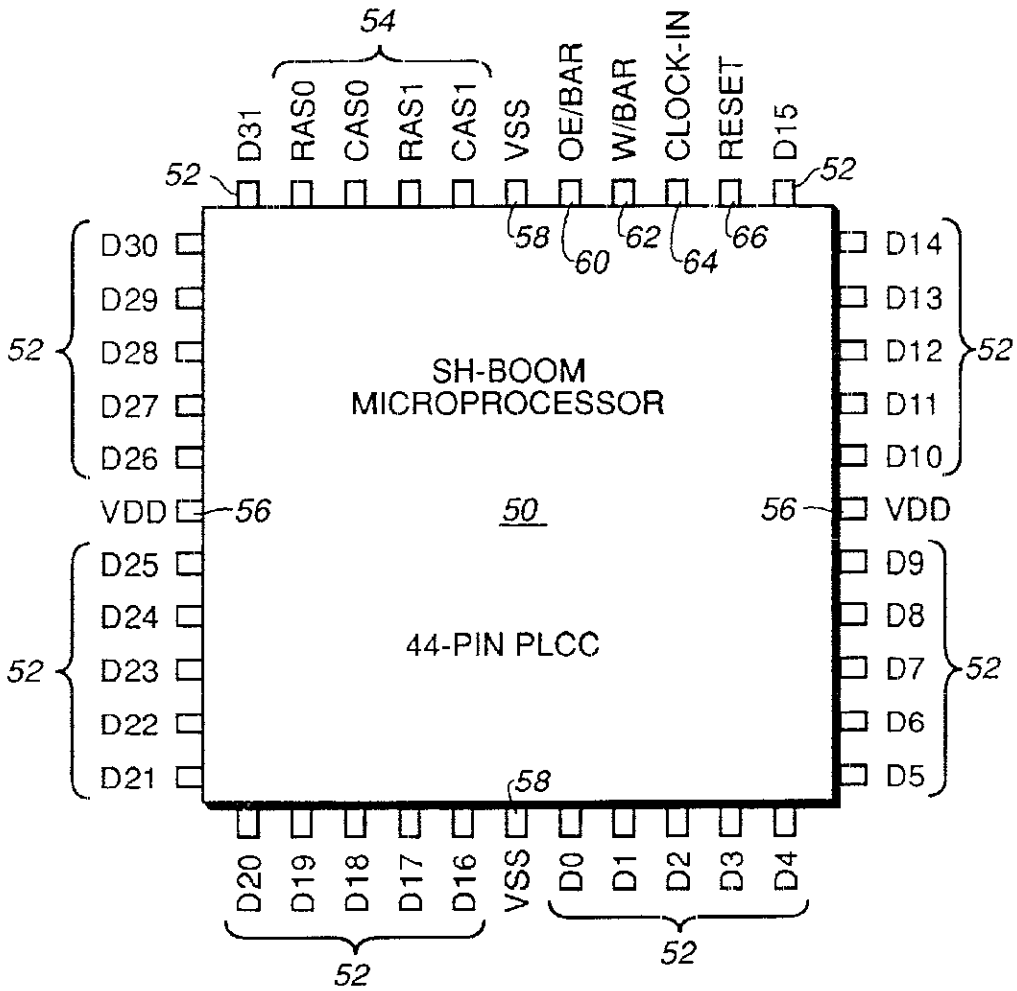

54
D31
RAS0
CAS0
RAS1
CAS1
VSS
OE/BAR
W/BAR
CLOCK-IN
RESET
D15
52
58
60
62
64
66
D30
D29
D28
D27
D26
VDD
D25
D24
D23
D22
D21
D14
D13
D12
D11
D10
VDD
D9
D8
D7
D6
D5
SH-BOOM
MICROPROCESSOR
50
44-PIN PLCC
56
D20
D19
D18
D17
D16
VSS
D0
D1
D2
D3
D4

FIG._1

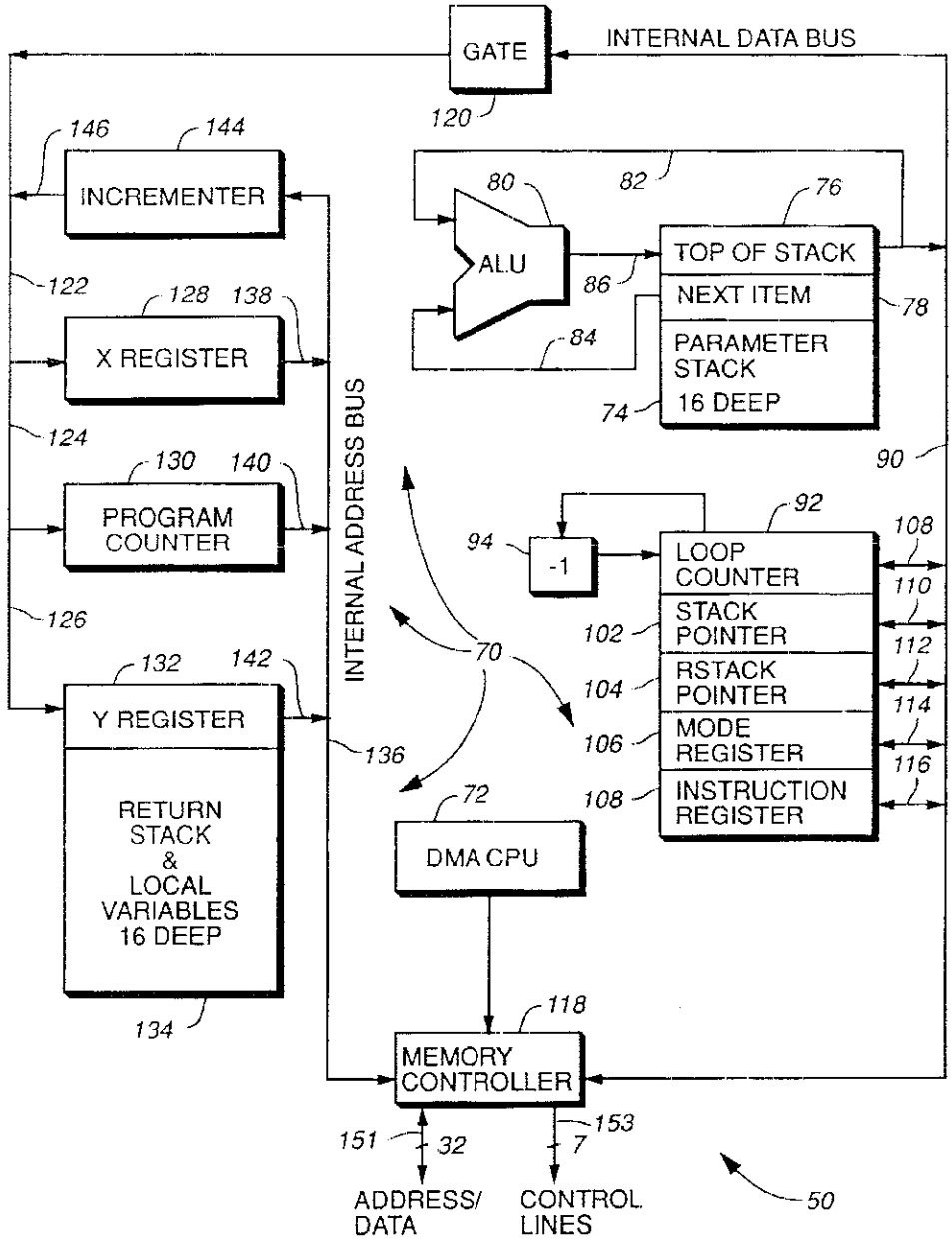

GATE
INTERNAL DATA BUS
120
146
144
INCREMENTER
80
82
76
ALU
TOP OF STACK
NEXT ITEM
PARAMETER STACK
16 DEEP
86
84
78
74
122
128
138
X REGISTER
124
130
140
PROGRAM COUNTER
90
92
94
-1
LOOP COUNTER
STACK POINTER
RSTACK POINTER
MODE REGISTER
INSTRUCTION REGISTER
108
110
112
114
116
102
104
106
108
126
132
142
Y REGISTER
RETURN STACK & LOCAL VARIABLES 16 DEEP
134
INTERNAL ADDRESS BUS
70
136
72
DMA CPU
118
MEMORY CONTROLLER
151
32
153
7
ADDRESS/ DATA
CONTROL LINES
50

FIG._2

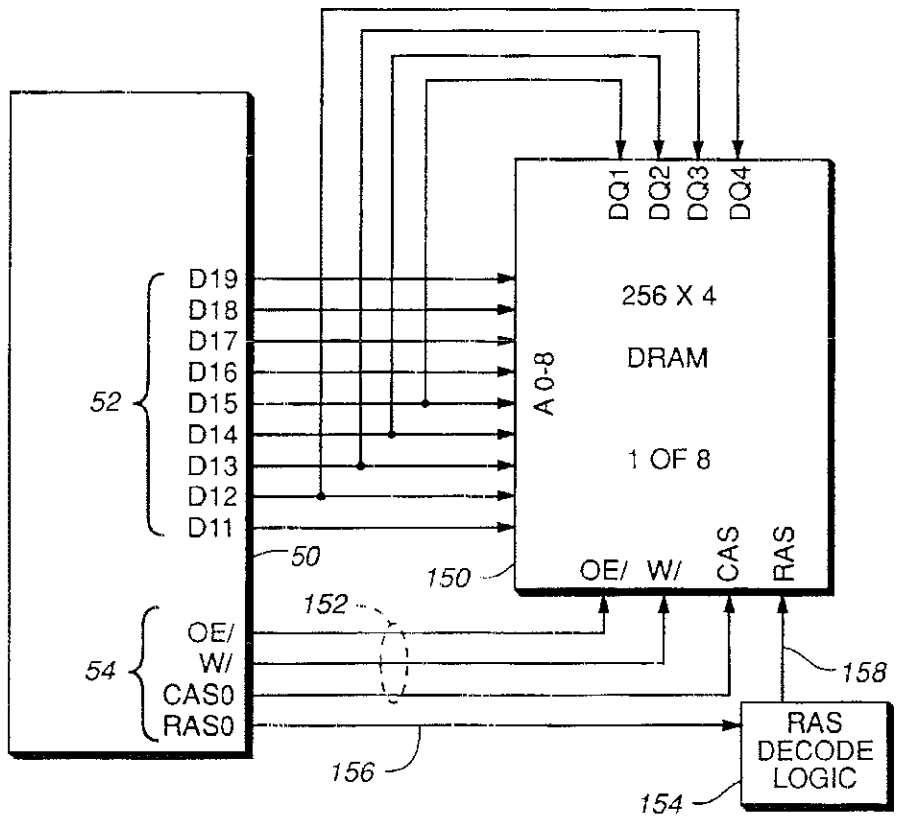

DQ1
DQ2
DQ3
DQ4
256 X 4
DRAM
1 OF 8
A 0-8
OE/
W/
CAS
RAS
150
D19
D18
D17
D16
D15
D14
D13
D12
D11
52
50
152
OE/
W/
CAS0
RAS0
54
158
RAS
DECODE
LOGIC
154
156

FIG._3

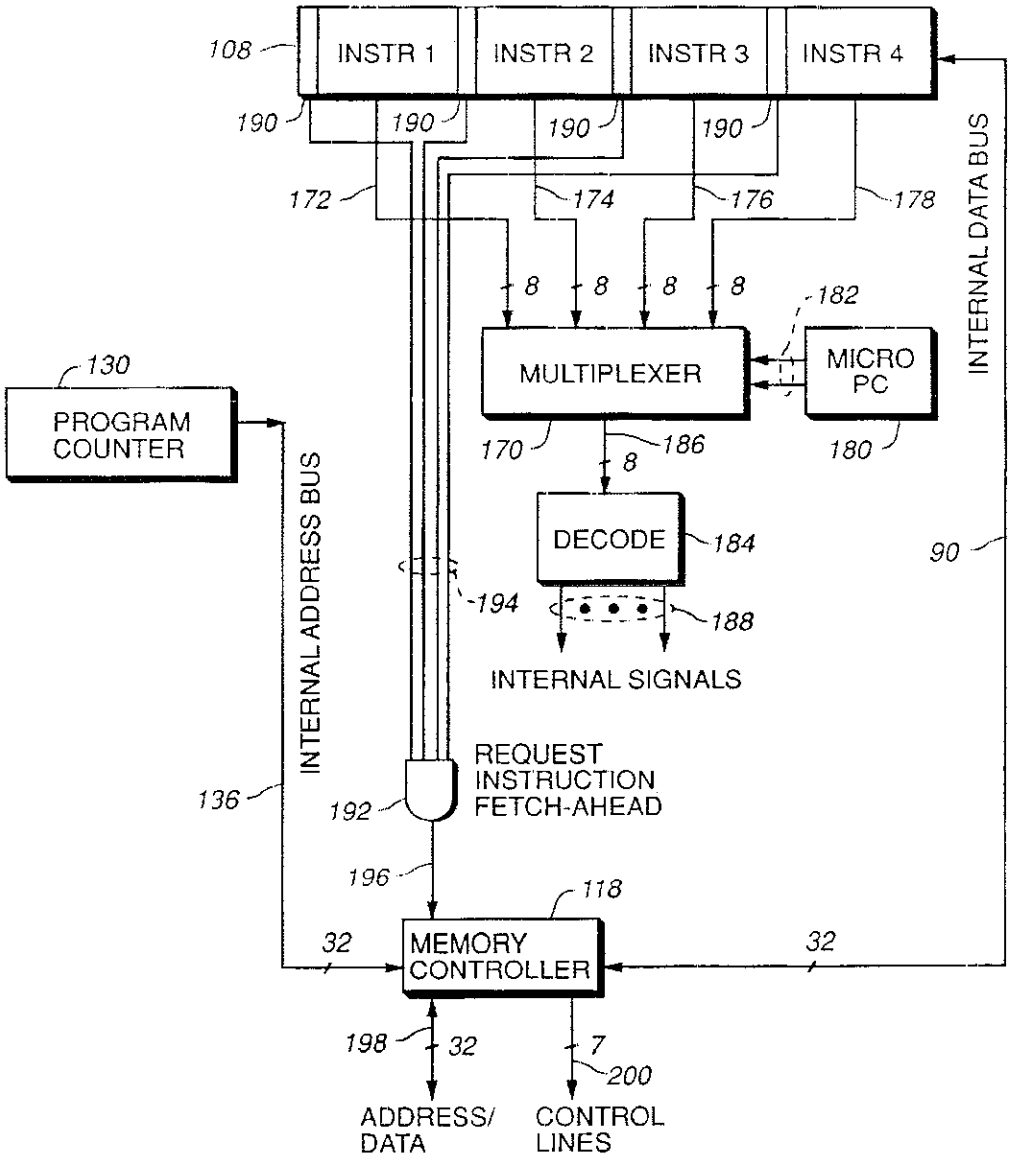

108
INSTR 1
INSTR 2
INSTR 3
INSTR 4
190
190
190
190
172
174
176
178
8
8
8
8
182
MULTIPLEXER
MICRO PC
INTERNAL DATA BUS
130
PROGRAM COUNTER
170
186
180
8
DECODE
184
90
194
188
INTERNAL SIGNALS
INTERNAL ADDRESS BUS
REQUEST INSTRUCTION FETCH-AHEAD
136
192
196
118
32
MEMORY CONTROLLER
32
198
32
7
200
ADDRESS/ DATA
CONTROL LINES

FIG._4

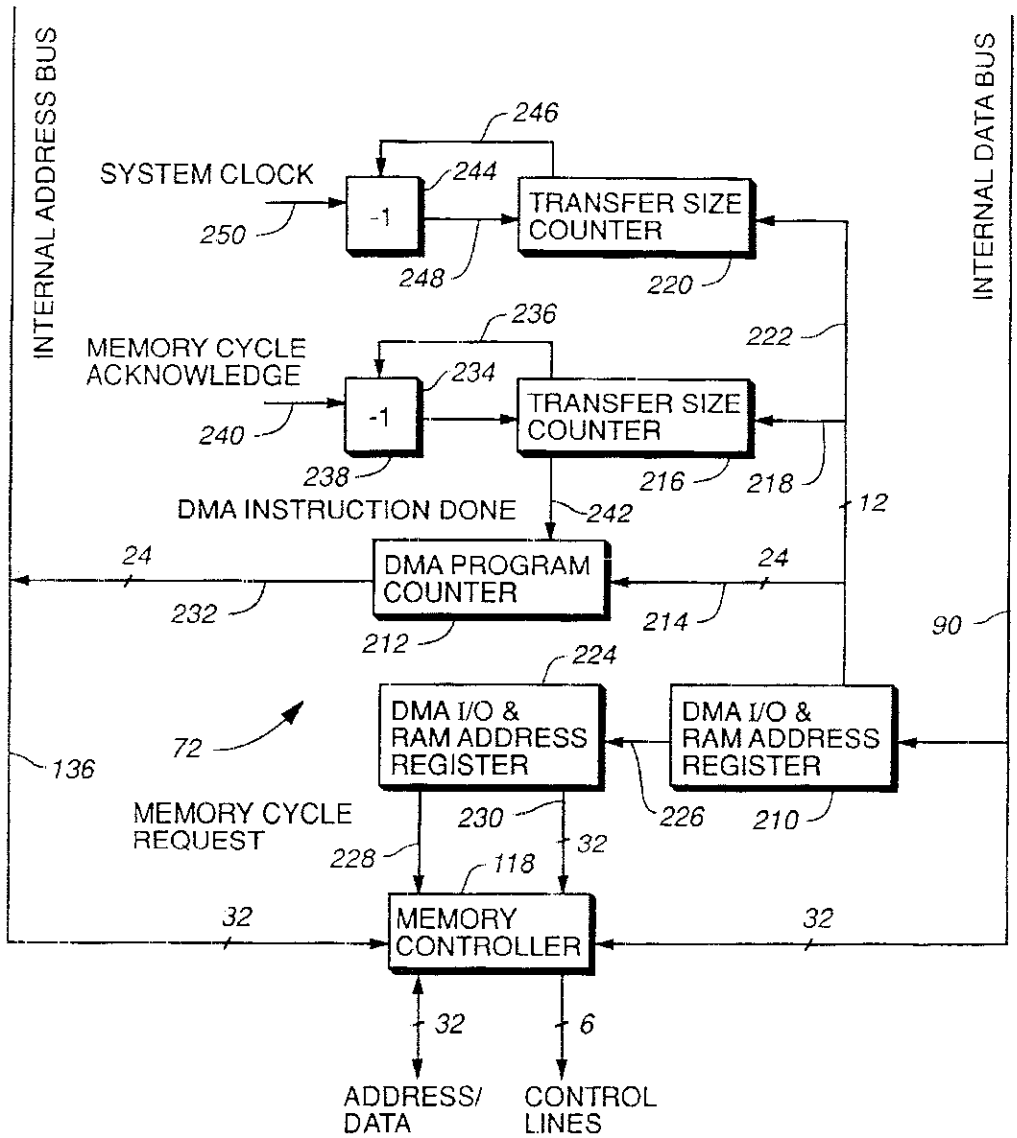

INTERNAL ADDRESS BUS
INTERNAL DATA BUS
SYSTEM CLOCK
250
-1
244
246
248
TRANSFER SIZE COUNTER
220
MEMORY CYCLE ACKNOWLEDGE
240
-1
234
236
238
TRANSFER SIZE COUNTER
216
218
222
12
DMA INSTRUCTION DONE
242
24
232
DMA PROGRAM COUNTER
212
24
214
90
224
72
136
DMA I/O & RAM ADDRESS REGISTER
DMA I/O & RAM ADDRESS REGISTER
226
210
MEMORY CYCLE REQUEST
228
230
32
118
32
MEMORY CONTROLLER
32
32
6
ADDRESS/ DATA
CONTROL LINES

FIG._5

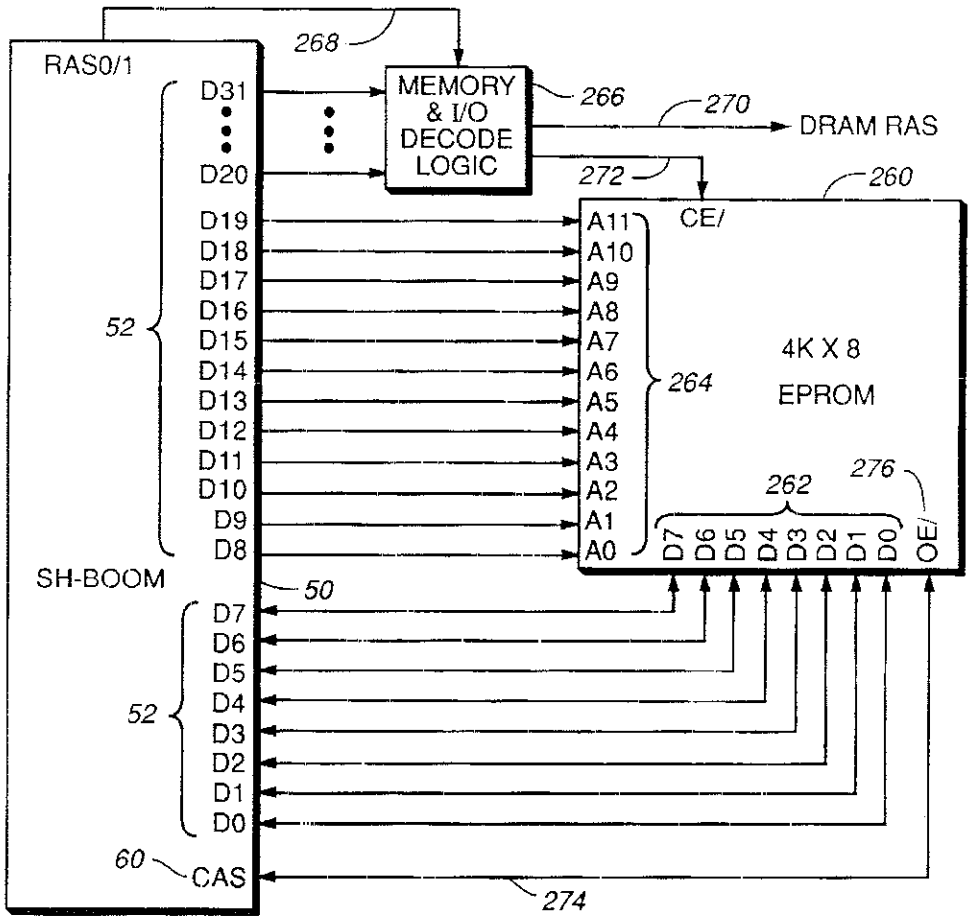

268
RAS0/1
D31
D20
MEMORY & I/O DECODE LOGIC
266
270
DRAM RAS
272
260
CE/
D19
D18
D17
D16
D15
D14
D13
D12
D11
D10
D9
D8
52
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
264
4K X 8
EPROM
262
276
D7
D6
D5
D4
D3
D2
D1
D0
OE/
SH-BOOM
50
D7
D6
D5
D4
D3
D2
D1
D0
52
60
CAS
274

FIG._6

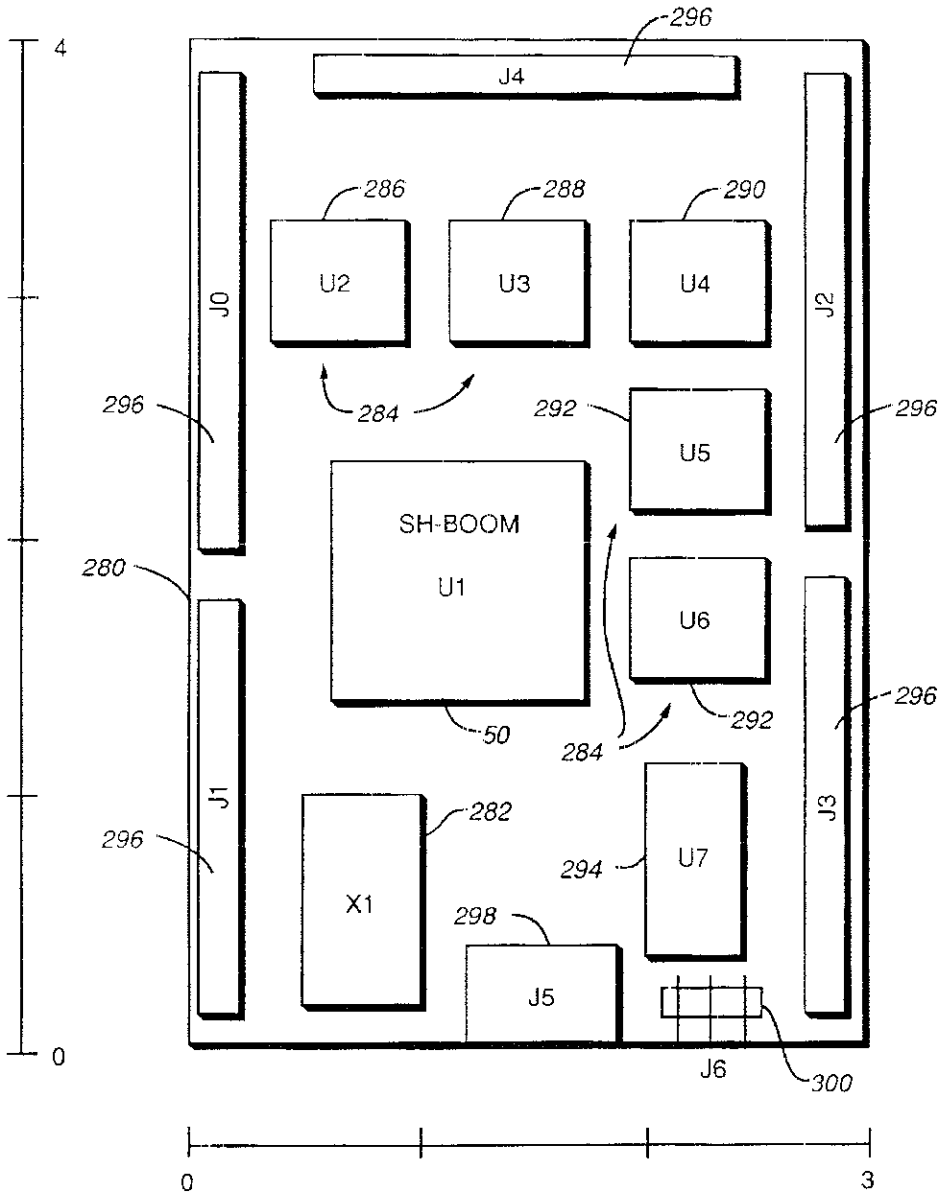

296
J4
286
288
290
U2
U3
U4
J0
J2
296
284
292
296
U5
SH-BOOM
U1
280
U6
296
50
292
284
J1
J3
296
282
294
U7
X1
298
J5
J6
300
4
0
0
3

FIG._7

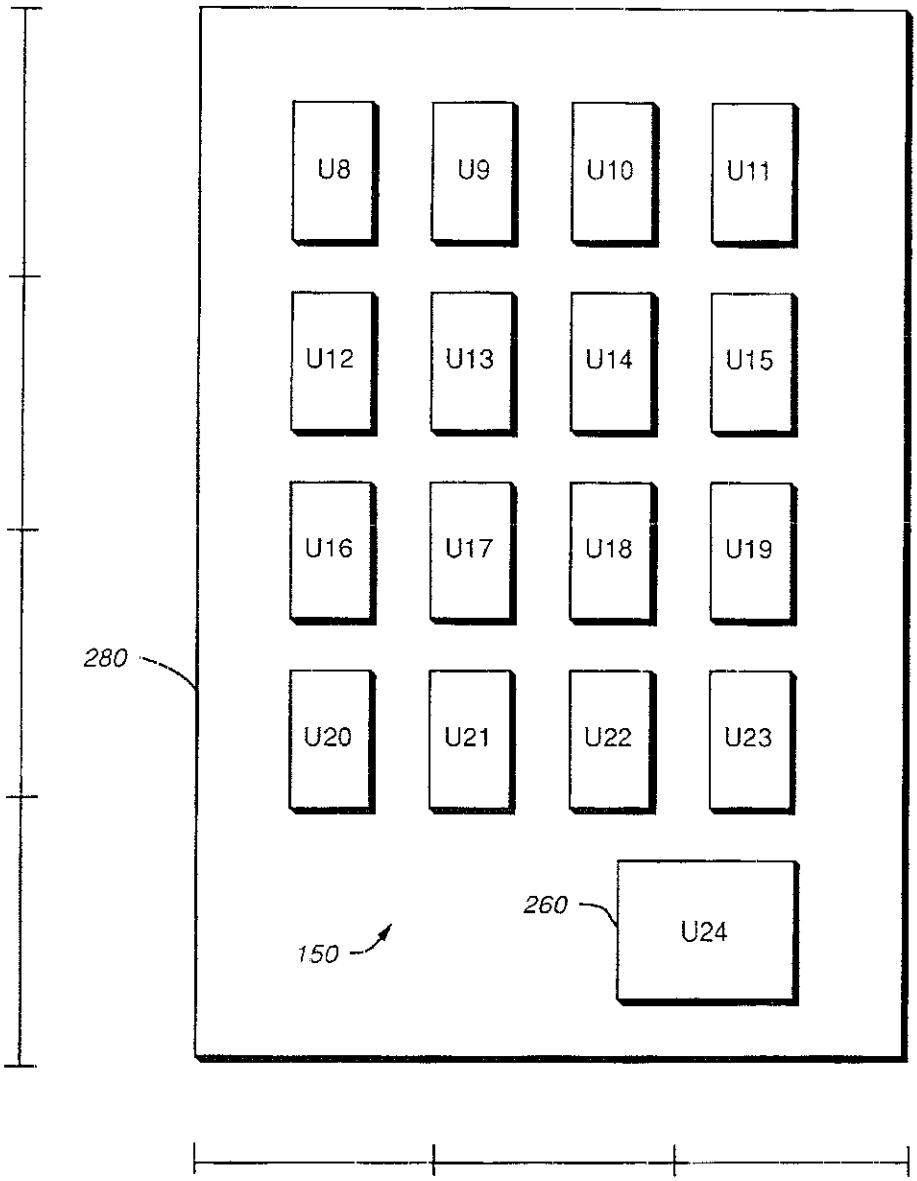

U8
U9
U10
U11
U12
U13
U14
U15
U16
U17
U18
U19
U20
U21
U22
U23
U24
280
260
150

FIG._8

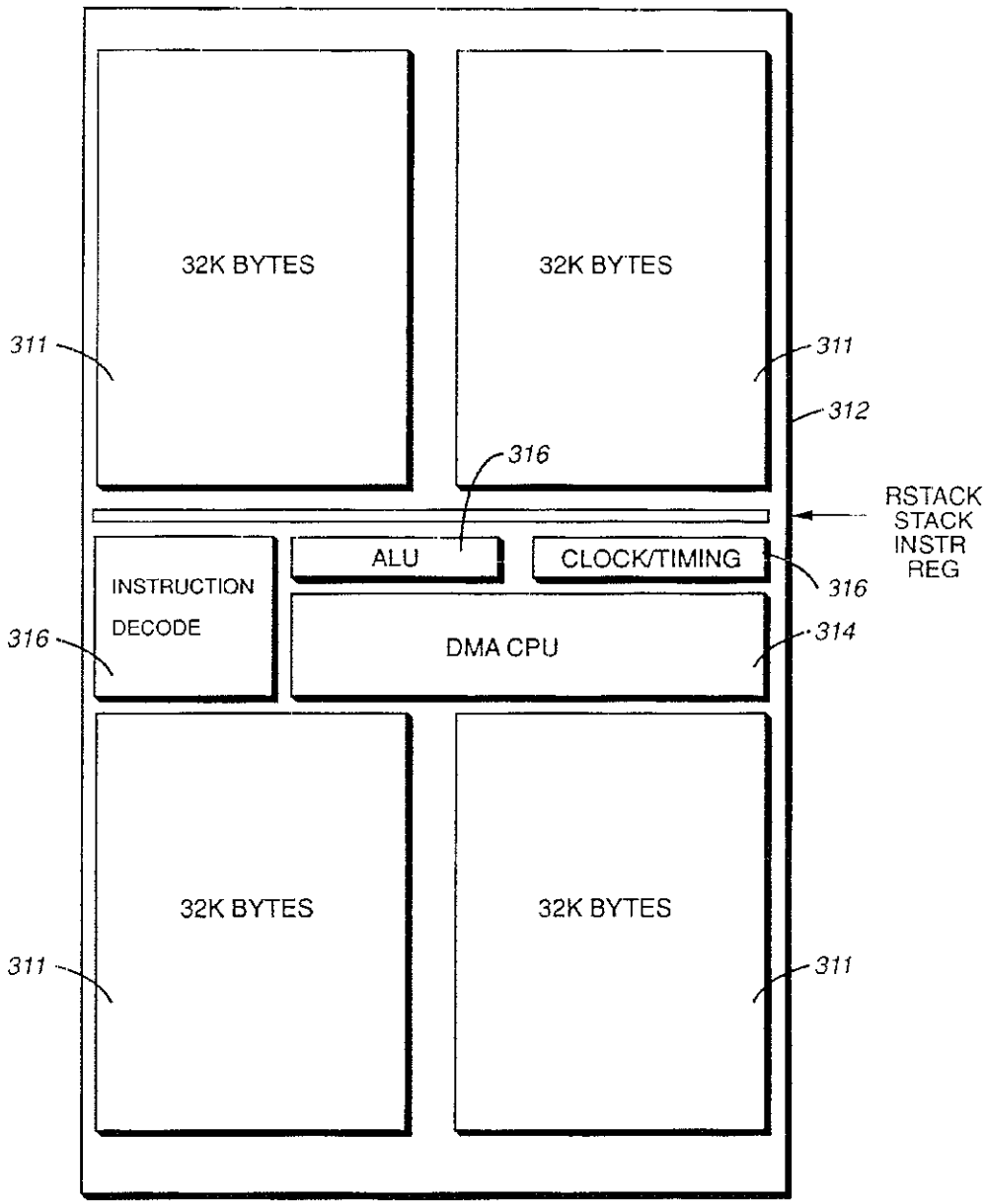

32K BYTES
32K BYTES
311
311
312
316
RSTACK
STACK
INSTR
REG
ALU
CLOCK/TIMING
316
INSTRUCTION
DECODE
316
DMA CPU
314
32K BYTES
32K BYTES
311
311

FIG._9

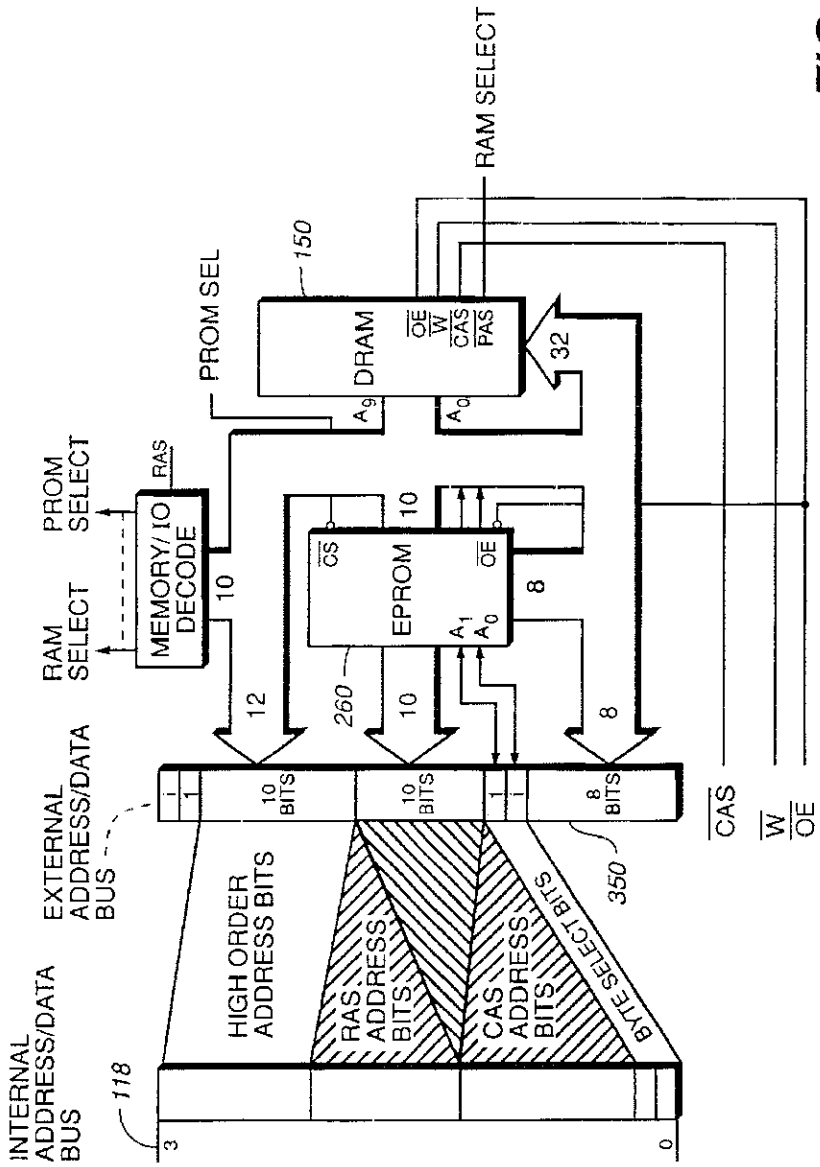
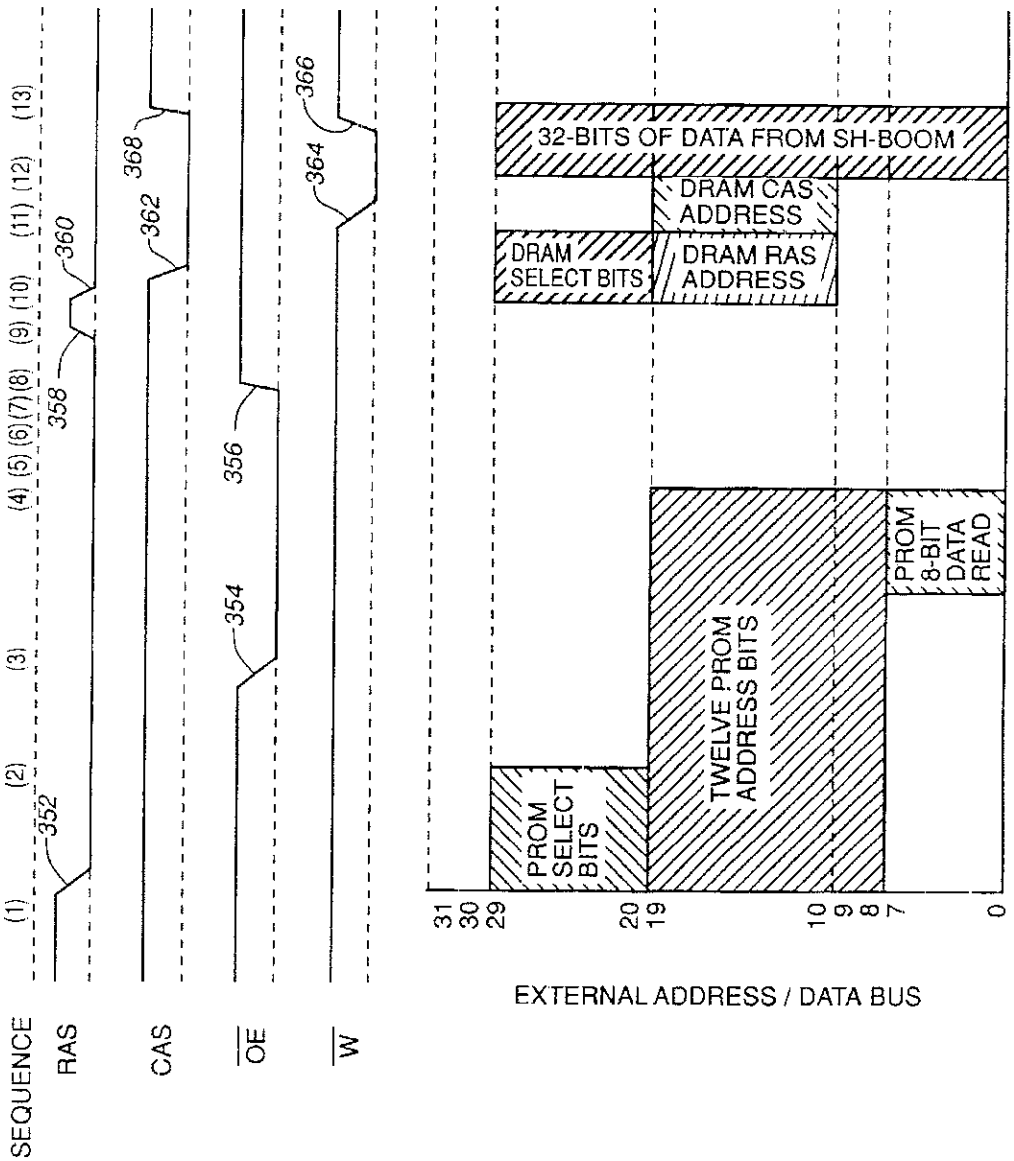

RAM SELECT
PROM SEL
150
DRAM
OE
W
CAS
PAS
32
A9
A0
PROM SELECT
RAS
MEMORY/IO DECODE
10
RAM SELECT
CS
EPROM
OE
A1
A0
10
8
260
12
10
8
EXTERNAL ADDRESS/DATA BUS
1
1
10 BITS
10 BITS
1
1
8 BITS
CAS
W
OE
350
HIGH ORDER ADDRESS BITS
RAS ADDRESS BITS
CAS ADDRESS BITS
BYTE SELECT BITS
INTERNAL ADDRESS/DATA BUS
118
3
0

FIG._10

FIG._11


SEQUENCE
RAS
CAS
OE
W
(1)
(2)
(3)
(4) (5) (6) (7) (8)
(9) (10)
(11) (12)
(13)
352
354
356
358
360
362
364
366
368
32-BITS OF DATA FROM SH-BOOM
DRAM CAS ADDRESS
DRAM SELECT BITS
DRAM RAS ADDRESS
PROM SELECT BITS
TWELVE PROM ADDRESS BITS
PROM 8-BIT DATA READ
31 30 29 20 19 10 9 8 7 0
EXTERNAL ADDRESS / DATA BUS

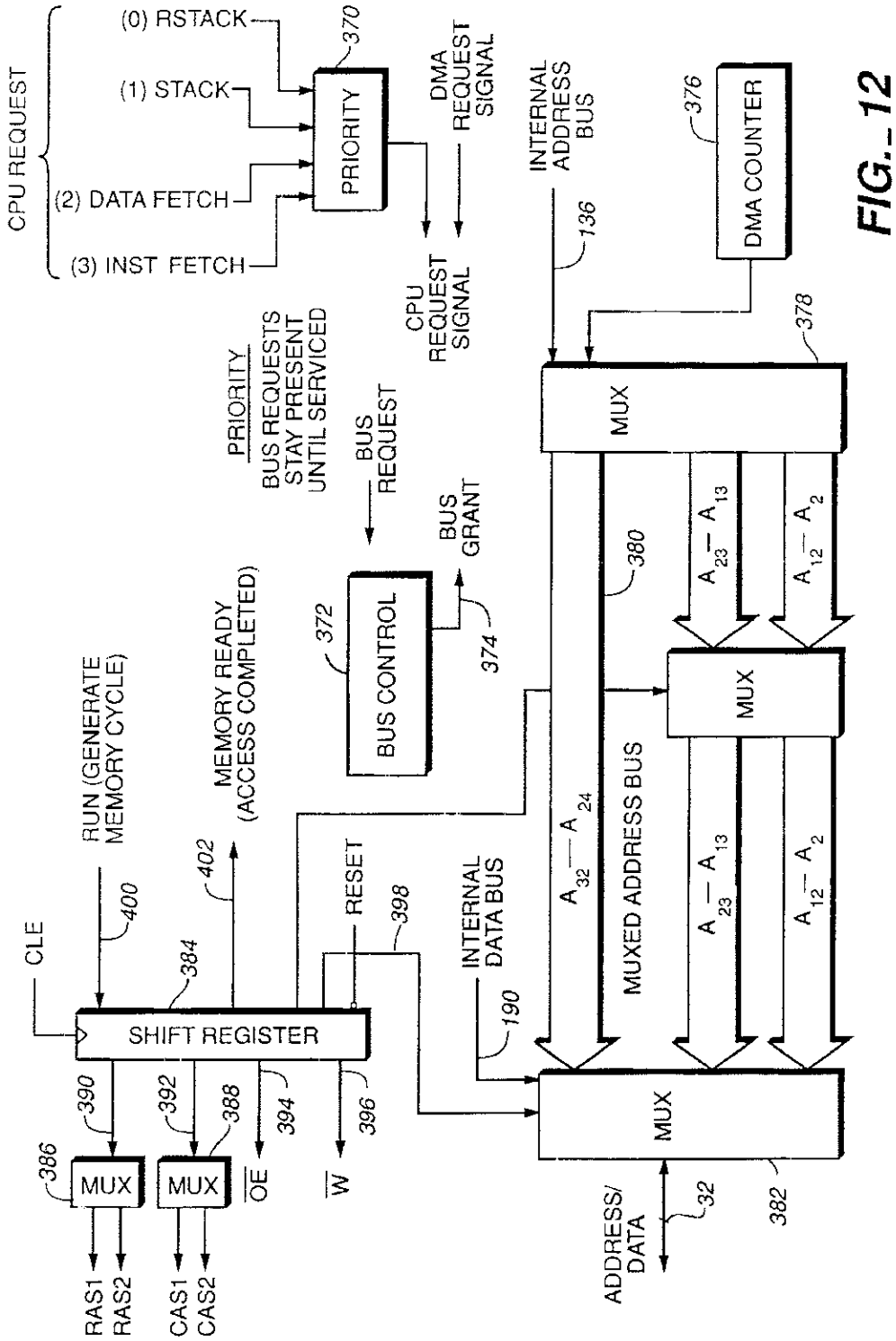

CPU REQUEST
(0) RSTACK
(1) STACK
(2) DATA FETCH
(3) INST FETCH
370
PRIORITY
DMA REQUEST SIGNAL
CPU REQUEST SIGNAL
INTERNAL ADDRESS BUS
136
376
DMA COUNTER
378
MUX
PRIORITY
BUS REQUESTS STAY PRESENT UNTIL SERVICED
BUS REQUEST
BUS GRANT
372
BUS CONTROL
374
380
A23 — A13
A12 — A2
MUX
MEMORY READY (ACCESS COMPLETED)
RUN (GENERATE MEMORY CYCLE)
400
402
RESET
398
CLE
384
INTERNAL DATA BUS
190
A32 — A24
MUXED ADDRESS BUS
SHIFT REGISTER
390
392
388
394
396
386
MUX
MUX
OE
W
MUX
RAS1
RAS2
CAS1
CAS2
ADDRESS/ DATA
32
382

FIG._12

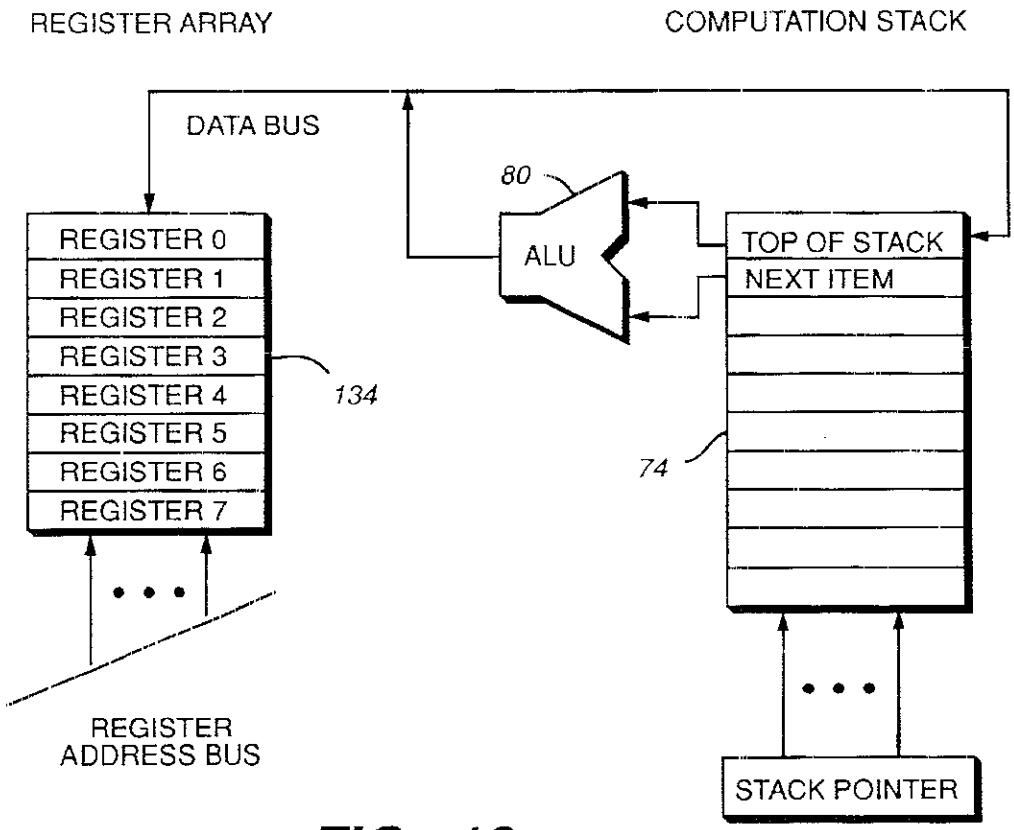

REGISTER ARRAY
COMPUTATION STACK
DATA BUS
80
ALU
REGISTER 0
REGISTER 1
REGISTER 2
REGISTER 3
REGISTER 4
REGISTER 5
REGISTER 6
REGISTER 7
134
TOP OF STACK
NEXT ITEM
74
REGISTER
ADDRESS BUS
STACK POINTER

FIG._13

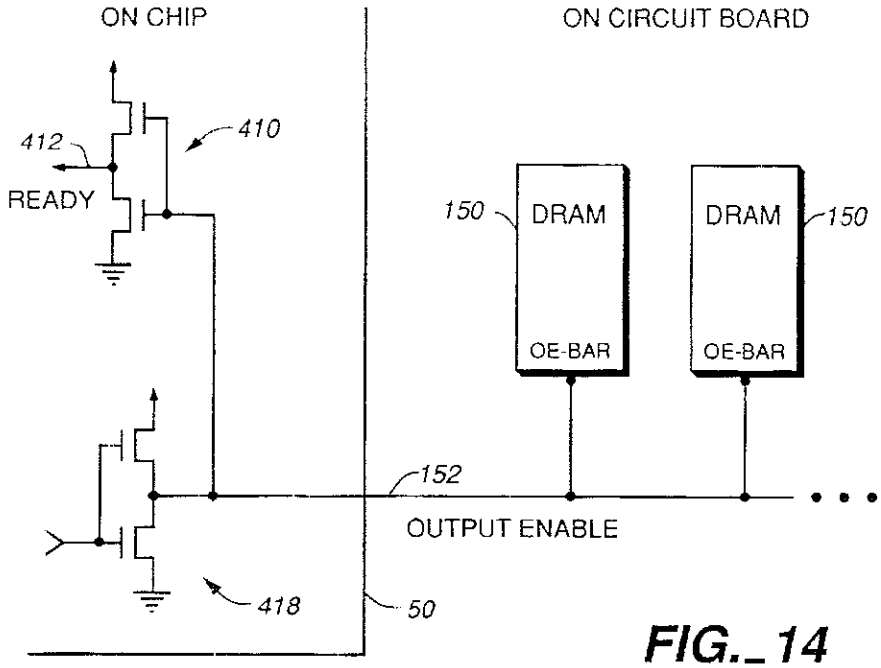

ON CHIP
ON CIRCUIT BOARD
412
410
READY
150
DRAM
OE-BAR
DRAM
OE-BAR
150
152
OUTPUT ENABLE
418
50

FIG._14

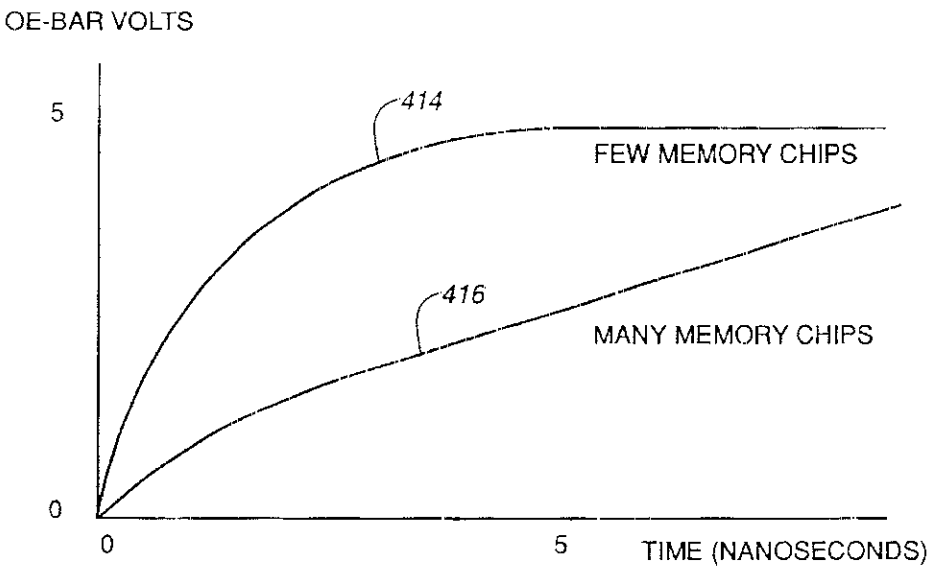

OE-BAR VOLTS
5
0
414
FEW MEMORY CHIPS
416
MANY MEMORY CHIPS
0
5
TIME (NANOSECONDS)

FIG._15

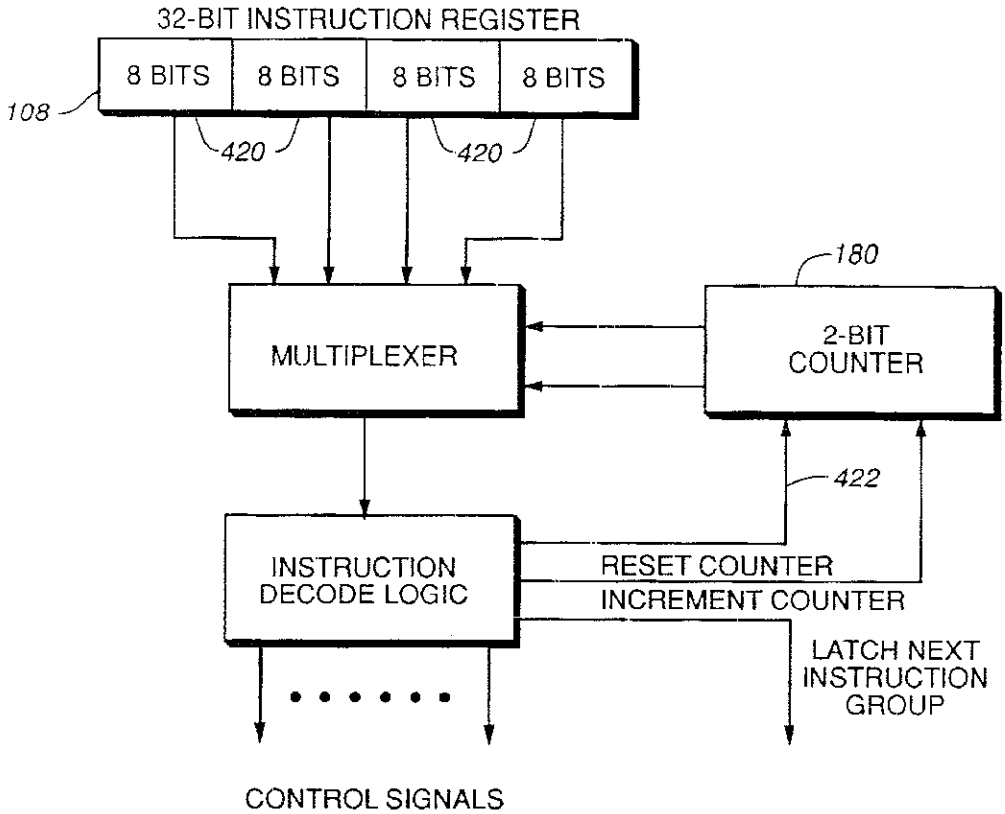

32-BIT INSTRUCTION REGISTER
8 BITS
8 BITS
8 BITS
8 BITS
108
420
420
180
MULTIPLEXER
2-BIT COUNTER
422
INSTRUCTION DECODE LOGIC
RESET COUNTER
INCREMENT COUNTER
LATCH NEXT INSTRUCTION GROUP
CONTROL SIGNALS

FIG._16

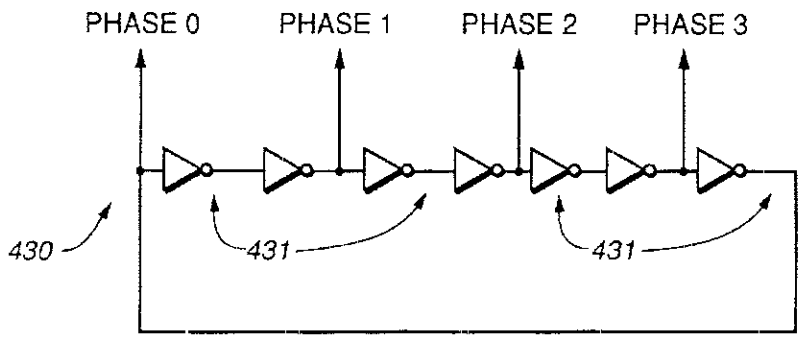

PHASE 0
PHASE 1
PHASE 2
PHASE 3
430
431
431

FIG._18

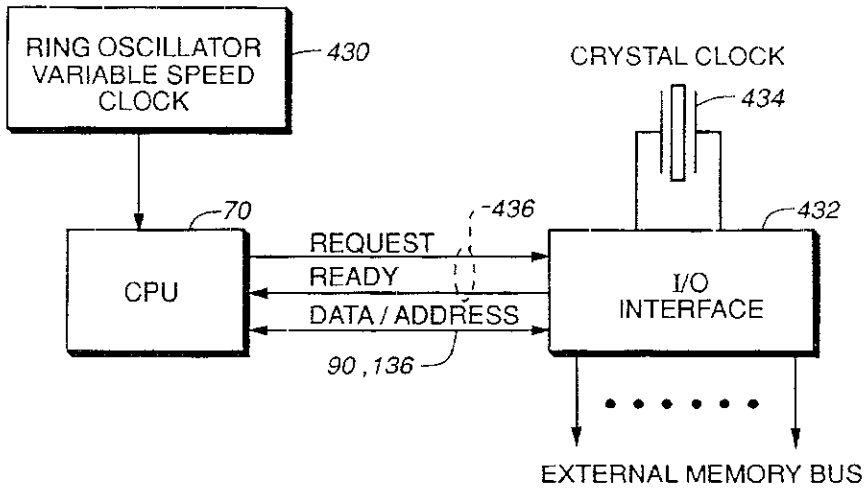

RING OSCILLATOR
VARIABLE SPEED
CLOCK
430
CRYSTAL CLOCK
434
70
436
432
CPU
REQUEST
READY
DATA / ADDRESS
I/O
INTERFACE
90 , 136
EXTERNAL MEMORY BUS

FIG._17

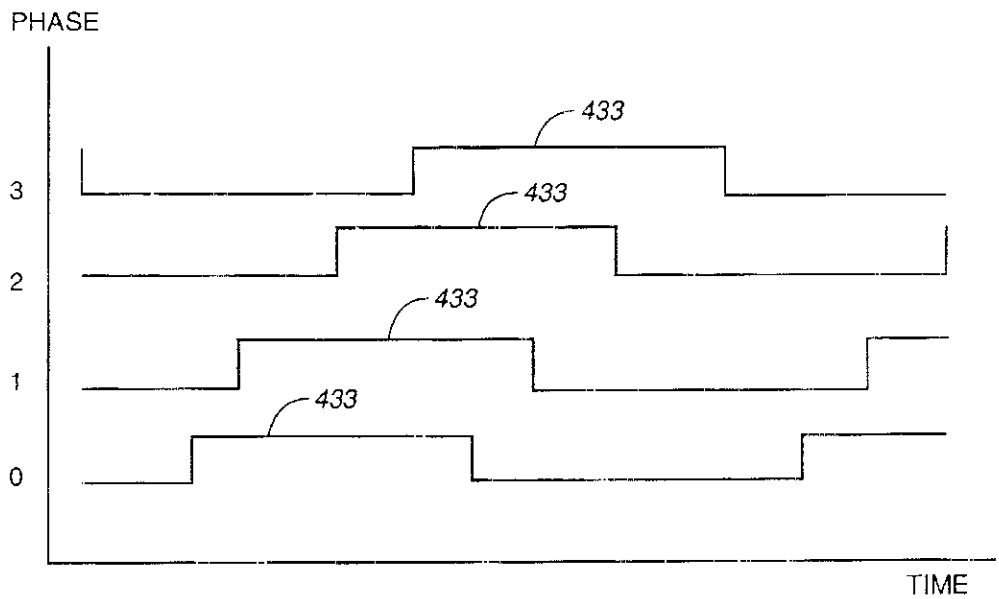

PHASE
3
2
1
0
433
433
433
433
TIME

FIG._19

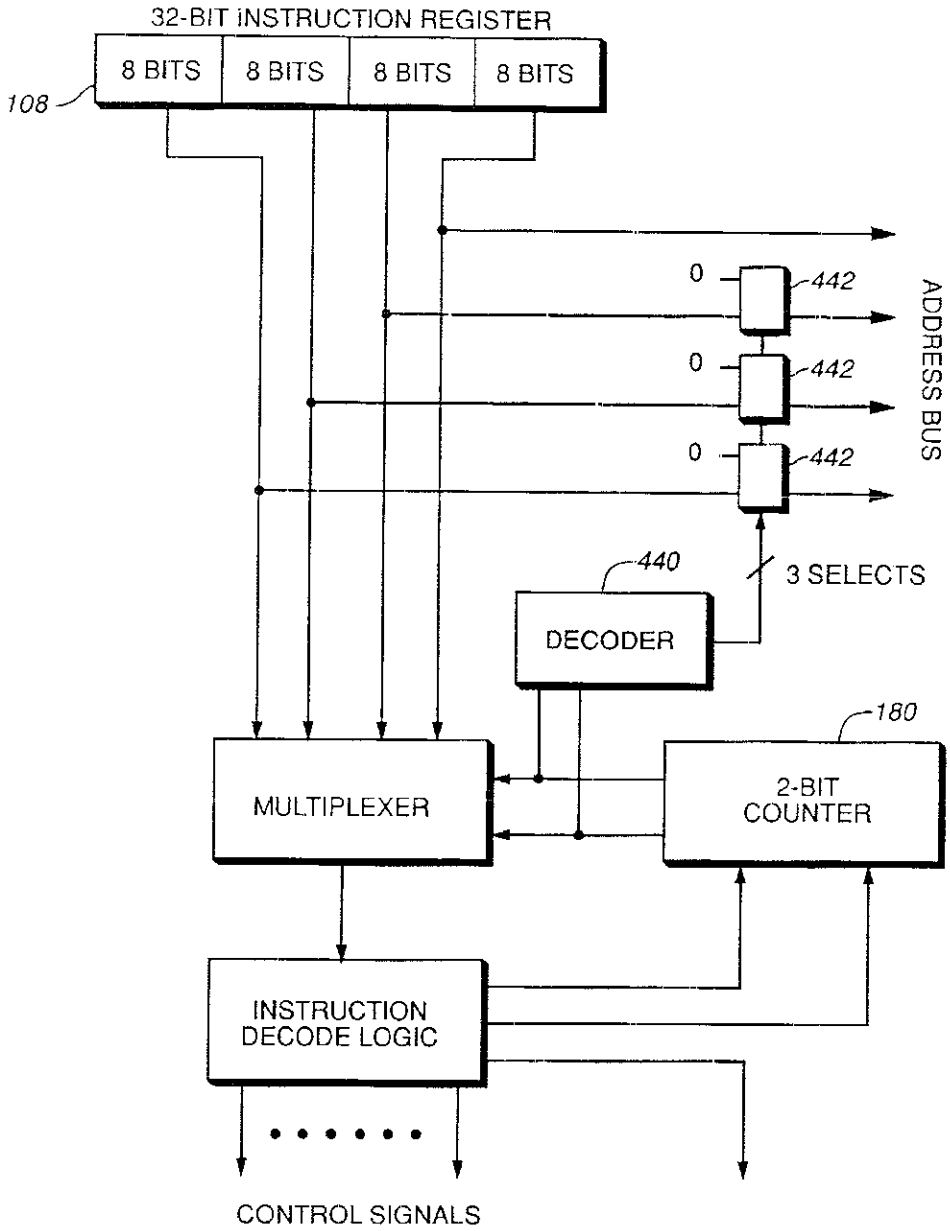

32-BIT INSTRUCTION REGISTER
8 BITS
8 BITS
8 BITS
8 BITS
108
0
442
0
442
0
442
ADDRESS BUS
440
3 SELECTS
DECODER
180
MULTIPLEXER
2-BIT
COUNTER
INSTRUCTION
DECODE LOGIC
CONTROL SIGNALS

FIG._20

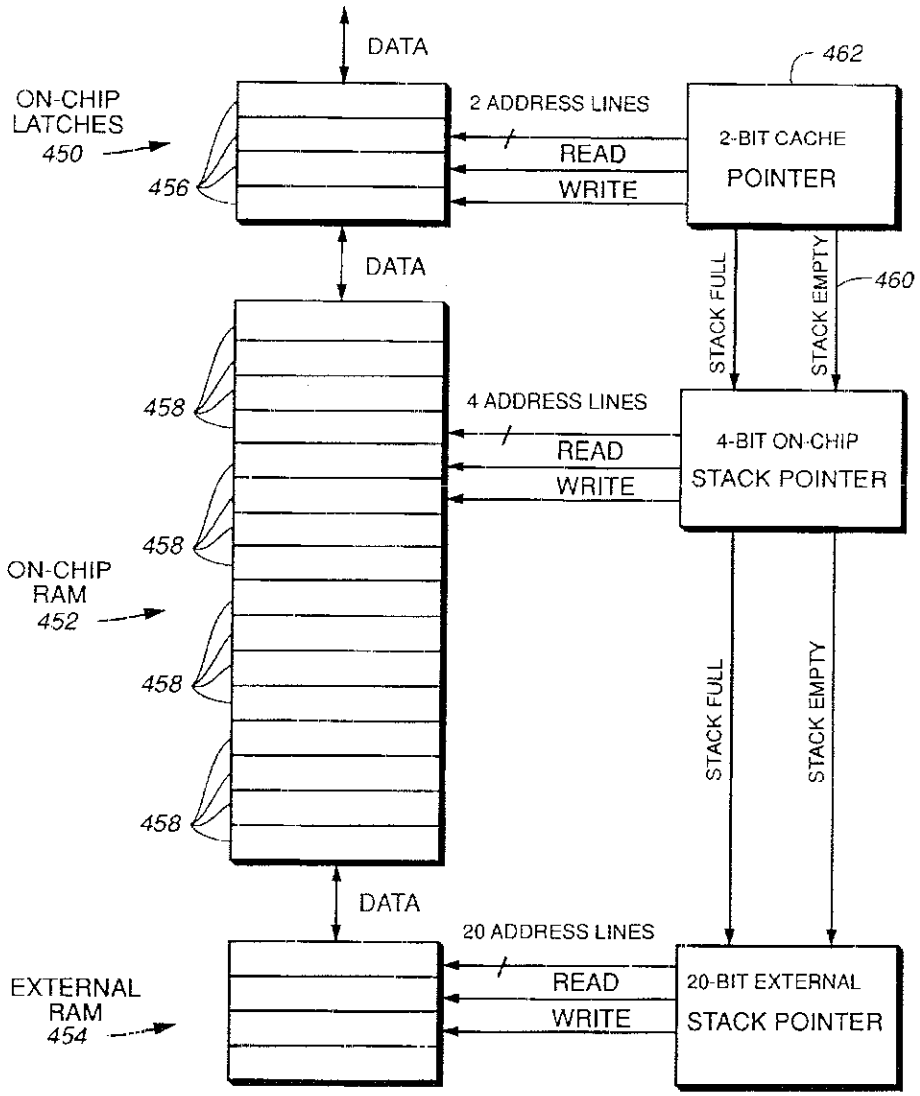

DATA
462
ON-CHIP LATCHES
450
2 ADDRESS LINES
READ
WRITE
2-BIT CACHE POINTER
456
DATA
STACK FULL
STACK EMPTY
460
458
4 ADDRESS LINES
READ
WRITE
4-BIT ON-CHIP STACK POINTER
458
ON-CHIP RAM
452
458
STACK FULL
STACK EMPTY
458
DATA
20 ADDRESS LINES
READ
WRITE
20-BIT EXTERNAL STACK POINTER
EXTERNAL RAM
454

FIG._21

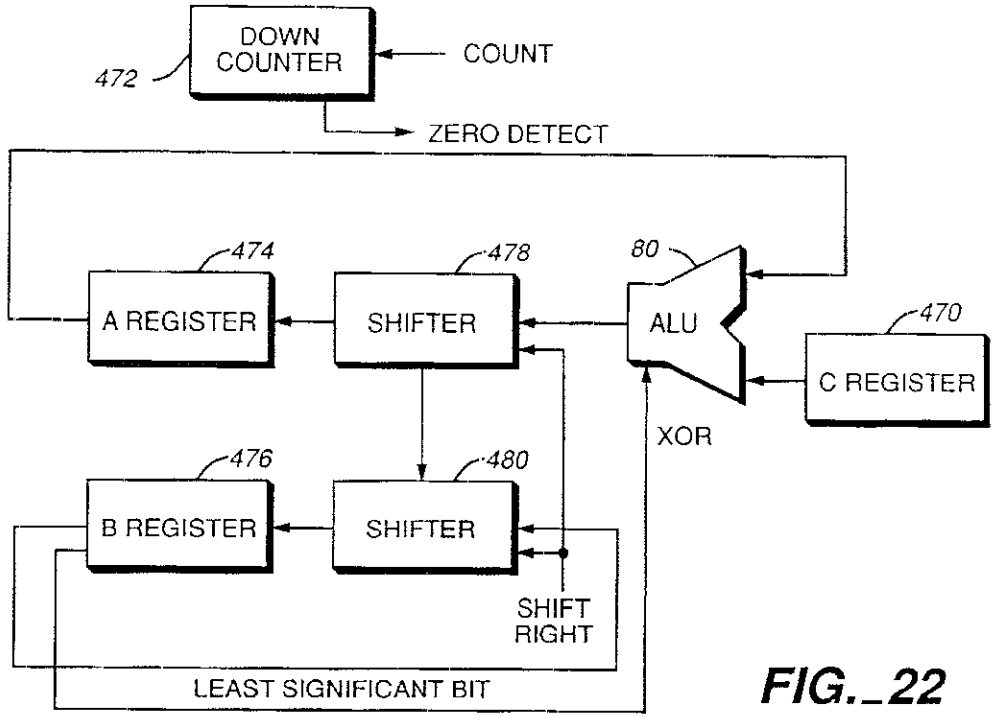

DOWN COUNTER
472
COUNT
ZERO DETECT
474
A REGISTER
478
SHIFTER
80
ALU
470
C REGISTER
XOR
476
B REGISTER
480
SHIFTER
SHIFT RIGHT
LEAST SIGNIFICANT BIT

FIG._22

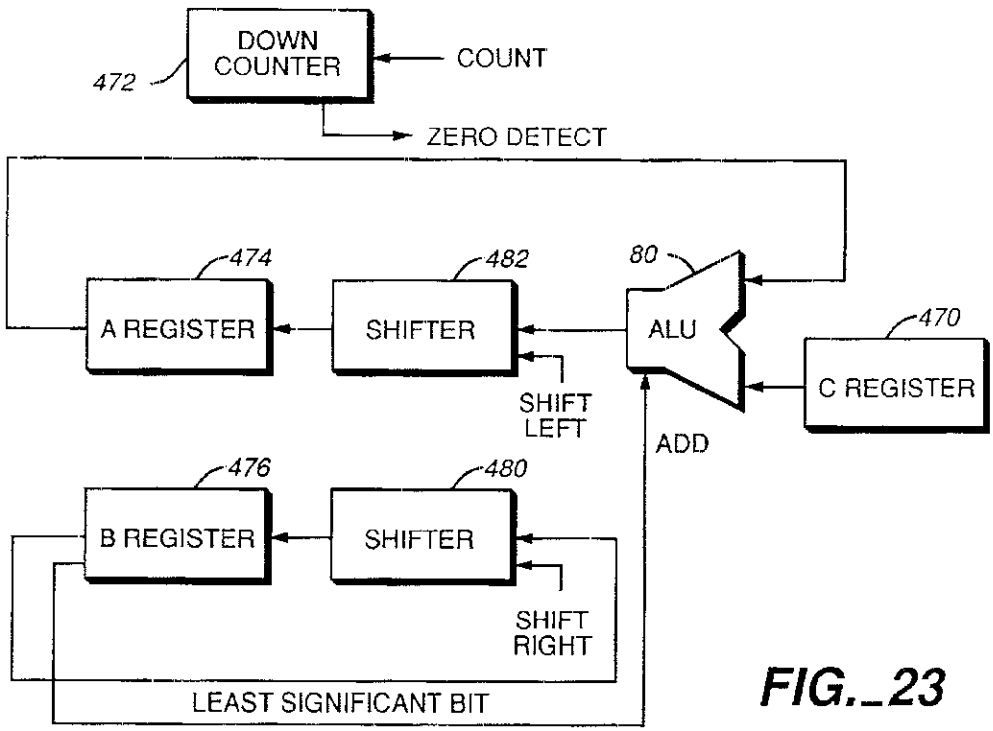

DOWN COUNTER
472
COUNT
ZERO DETECT
474
A REGISTER
482
SHIFTER
80
ALU
470
C REGISTER
SHIFT LEFT
ADD
476
B REGISTER
480
SHIFTER
SHIFT RIGHT
LEAST SIGNIFICANT BIT

FIG._23

HIGH PERFORMANCE MICROPROCESSOR HAVING VARIABLE SPEED SYSTEM CLOCK

This application is a divisional of U S patent application No. 08/484,918, filed Jun. 7. 1995, now U S Pat No 5,809,336 which is a divisional of U S patent application No. 07/389,334 filed Aug 3. 1989 now U S Pat No 5,982,231

BACKGROUND OF THE INVENTION

1 Field of the Invention

The present invention relates generally to a simplified, reduced instruction set computer (RISC) microprocessor More particularly it relates to such a microprocessor which is capable of performance levels of, for example, 20 million instructions per second (MIPS) at a price of, for example, 20 dollars

2 Description of the Prior Art

Since the invention of the microprocessor, improvements in its design have taken two different approaches In the first approach, a brute force gain in performance has been achieved through the provision of greater numbers of faster transistors in the microprocessor integrated circuit and an instruction set of increased complexity. This approach is exemplified by the Motorola 68000 and Intel 80X86 microprocessor families The trend in this approach is to larger die sizes and packages with hundreds of pinouts

More recently, it has been perceived that performance gains can be achieved through comparative simplicity, both in the microprocessor integrated circuit itself and in its instruction set. This second approach provides RISC microprocessors, and is exemplified by the Sun SPARC and the Intel 8960 microprocessors. However, even with this approach as conventionally practiced, the packages for the microprocessor are large, in order to accommodate the large number of pinouts that continue to be employed. A need therefore remains for further simplification of high performance microprocessors.

With conventional high performance microprocessors, fast static memories are required for direct connection to the microprocessors in order to allow memory accesses that are fast enough to keep up with the microprocessors Slower dynamic random access memories (DRAMs) are used with such microprocessors only in a hierarchical memory arrangement, with the static memories acting as a buffer between the microprocessors and the DRAMs The necessity to use static memories increases cost of the resulting systems.

Conventional microprocessors provide direct memory accesses (DMA) for system peripheral units through DMA controllers, which may be located on the microprocessor integrated circuit or provided separately. Such DMA controllers can provide routine handling of DMA requests and responses, but some processing by the main central processing unit (CPU) of the microprocessor is required

SUMMARY OF THE INVENTION

Accordingly, it is an object of this invention to provide a microprocessor with a reduced pin count and cost compared to conventional microprocessors

It is another object of the invention to provide a high performance microprocessor that can be directly connected to DRAMs without sacrificing microprocessor speed

It is a further object of the invention to provide a high performance microprocessor in which DMA does not require use of the main CPU during DMA requests and responses and which provides very rapid DMA response with predictable response times

The attainment of these and related objects may be achieved through use of the novel high performance low cost microprocessor herein disclosed The microprocessor integrated circuit includes a processing unit disposed upon an integrated circuit substrate In a preferred implementation the processing unit operates in accordance with a predefined sequence of program instructions stored within an instruction register A memory, capable of storing information provided by the processing unit and occupying a larger area of the integrated circuit substrate than the processing unit, is also provided within the microprocessor integrated circuit The memory may be implemented using, for example, dynamic or static random-access memory

The attainment of the foregoing and related objects, advantages and features of the invention should be more readily apparent to those skilled in the art, after review of the following more detailed description of the invention taken together with the drawings in which:

BRIEF DESCRIPTION OF THE DRAWINGS

FIG 1 is an external, plan view of an integrated circuit package incorporating a microprocessor in accordance with the invention

FIG 2 is a block diagram of a microprocessor in accordance with the invention

FIG 3 is a block diagram of a portion of a data processing system incorporating the microprocessor of FIGS 1 and 2.

FIG 4 is a more detailed block diagram of a portion of the microprocessor shown in FIG 2

FIG 5 is a more detailed block diagram of another portion of the microprocessor shown in FIG. 2

FIG 6 is a block diagram of another portion of the data processing system shown in part in FIG. 3 and incorporating the microprocessor of FIGS. 1–2 and 4–5

FIGS 7 and 8 are layout diagrams for the data processing system shown in part in FIGS 3 and 6

FIG. 9 is a layout diagram of a second embodiment of a microprocessor in accordance with the invention in a data processing system on a single integrated circuit

FIG 10 is a more detailed block diagram of a portion of the data processing system of FIGS 7 and 8

FIG 11 is a timing diagram useful for understanding operation of the system portion shown in FIG 12

FIG 12 is another more detailed block diagram of a further portion of the data processing system of FIGS 7 and 8

FIG 13 is a more detailed block diagram of a portion of the microprocessor shown in FIG 2

FIG 14 is a more detailed block and schematic diagram of a portion of the system shown in FIGS 3 and 7–8

FIG 15 is a graph useful for understanding operation of the system portion shown in FIG 14

FIG. 16 is a more detailed block diagram showing part of the system portion shown in FIG 4

FIG 17 is a more detailed block diagram of a portion of the microprocessor shown in FIG 2

FIG. 18 is a more detailed block diagram of part of the microprocessor portion shown in FIG 17

FIG 19 is a set of waveform diagrams useful for understanding operation of the part of the microprocessor portion shown in FIG 18

FIG. **20** is a more detailed block diagram showing another part of the system portion shown in FIG **4**

FIG. **21** is a more detailed block diagram showing another part of the system portion shown in FIG **4**

FIGS **22** and **23** are more detailed block diagrams showing another part of the system portion shown in FIG **4**

DETAILED DESCRIPTION OF THE INVENTION

OVERVIEW

The microprocessor of this invention is desirably implemented as a 32-bit microprocessor optimized for:

HIGH EXECUTION SPEED, and

LOW SYSTEM COST

In this embodiment, the microprocessor can be thought of as 20 MIPS for 20 dollars Important distinguishing features of the microprocessor are:

Uses low-cost commodity DYNAMIC RAMS to run 20 MIPS

4 instruction fetch per memory cycle

On-chip fast page-mode memory management

Runs fast without external cache

Requires few interfacing chips

Crams 32-bit CPU in 44 pin SOJ package

The instruction set is organized so that most operations can be specified with 8-bit instructions Two positive products of this philosophy are:

Programs are smaller,

Programs can execute much faster

The bottleneck in most computer systems is the memory bus The bus is used to fetch instructions and fetch and store data The ability to fetch four instructions in a single memory bus cycle significantly increases the bus availability to handle data

Turning now to the drawings, more particularly to FIG **1**, there is shown a packaged 32-bit microprocessor **50** in a 44-pin plastic leadless chip carrier. shown approximately 100 times its actual size of about 0.8 inch on a side The fact that the microprocessor **50** is provided as a 44-pin package represents a substantial departure from typical microprocessor packages, which usually have about 200 input/output (I/O) pins The microprocessor **50** is rated at 20 million instructions per second (MIPS) Address and data lines **52**. also labelled **D0**–**D31**. are shared for addresses and data without speed penalty as a result of the manner in which the microprocessor **50** operates., as will be explained below

DYNAMIC RAM

In addition to the low cost 44-pin package. another unusual aspect of the high performance microprocessor **50** is that it operates directly with dynamic random access memories (DRAMs), as shown by row address strobe (RAS) and column address strobe (CAS) I/O pins **54** The other I/O pins for the microprocessor **50** include VDD pins **56**, VSS pins **58**, output enable pin **60**. write pin **62**, clock pin **64** and reset pin **66**.

All high speed computers require high speed and expensive memory to keep up The highest speed static RAM memories cost as much as ten times as much as slower dynamic RAMs This microprocessor has been optimized to use low-cost dynamic RAM in high-speed page-mode Page-mode dynamic RAMs offer static RAM performance without the cost penalty For example. low-cost 85 nsec dynamic RAMs access at 25 nsec when operated in fast page-mode. Integrated fast page-mode control on the microprocessor chip simplifies system interfacing and results in a faster system

Details of the microprocessor **50** are shown in FIG **2** The microprocessor **50** includes a main central processing unit (CPU) **70** and a separate direct memory access (DMA) CPU **72** in a single integrated circuit making up the microprocessor **50** The main CPU **70** has a first **16** deep push down stack **74**, which has a top item register **76** and a next item register **78**. respectively connected to provide inputs to an arithmetic logic unit (ALU) **80** by lines **82** and **84** An output of the ALU **80** is connected to the top item register **76** by line **86** The output of the top item register at **82** is also connected by line **88** to an internal data bus **90**

A loop counter **92** is connected to a decrementer **94** by lines **96** and **98** The loop counter **92** is bidirectionally connected to the internal data bus **90** by line **100** Stack pointer **102**, return stack pointer **104**, mode register **106** and instruction register **108** are also connected to the internal data bus **90** by lines **110**. **112**, **114** and **116**. respectively The internal data bus **90** is connected to memory controller **118** and to gate **120** The gate **120** provides inputs on lines **122**, **124**, and **126** to X register **128**, program counter **130** and Y register **132** of return push down stack **134** The X register **128**, program counter **130** and Y register **132** provide outputs to internal address bus **136** on lines **138**, **140** and **142** The internal address bus provides inputs to the memory controller **118** and to an incrementer **144** The incrementer **144** provides inputs to the X register, program counter and Y register via lines **146**, **122**, **124** and **126** The DMA CPU **72** provides inputs to the memory controller **118** on line **148** The memory controller **118** is connected to a RAM (not shown) by address/data bus **151** and control lines **153**

FIG **2** shows that the microprocessor **50** has a simple architecture Prior art RISC microprocessors are substantially more complex in design For example, the SPARC RISC microprocessor has three times the gates of the microprocessor **50**, and the Intel 8960 RISC microprocessor has 20 times the gates of the microprocessor **50** The speed of this microprocessor is in substantial part due to this simplicity The architecture incorporates push down stacks and register write to achieve this simplicity.

The microprocessor **50** incorporates an I/O that has been tuned to make heavy use of resources provided on the integrated circuit chip On chip latches allow use of the same I/O circuits to handle three different things: column addressing, row addressing and data. with a slight to non-existent speed penalty This triple bus multiplexing results in fewer buffers to expand, fewer interconnection lines fewer I/O pins and fewer internal buffers.

The provision of on-chip DRAM control gives a performance equal to that obtained with the use of static RAMs. As a result, memory is provided at 1/4 the system cost of static RAM used in most RISC systems

The microprocessor **50** fetches 4 instructions per memory cycle; the instructions are in an 8-bit format. and this is a 32-bit microprocessor System speed is therefore 4 times the memory bus bandwidth. This ability enables the microprocessor to break the Von Neumann bottleneck of the speed of getting the next instruction This mode of operation is possible because of the use of a push down stack and register array The push down stack allows the use of implied addresses, rather than the prior art technique of explicit addresses for two sources and a destination

Most instructions execute in 20 nanoseconds in the microprocessor **50**. The microprocessor can therefore execute instructions at 50 peak MIPS without pipeline delays This is a function of the small number of gates in the microprocessor **50** and the high degree of parallelism in the architecture of the microprocessor

FIG 3 shows how column and row addresses are multiplexed on lines D8–D14 of the microprocessor **50** for addressing DRAM **150** from I/O pins **52**. The DRAM **150** is one of eight, but only one DRAM **150** has been shown for clarity. As shown, the lines D11–D18 are respectively connected to row address inputs A0–A8 of the DRAM **150**. Additionally, lines D12–D15 are connected to the data inputs DQ1–DQ4 of the DRAM **150**. The output enable, write and column address strobe pins **54** are respectively connected to the output enable, write and column address strobe inputs of the DRAM **150** by lines **152**. The row address strobe pin **54** is connected through row address strobe decode logic **154** to the row address strobe input of the DRAM **150** by lines **156** and **158**.

D0–D7 pins **52** (FIG 1) are idle when the microprocessor **50** is outputting multiplexed row and column addresses on D11D18 pins **52**. The D0–D7 pins **52** can therefore simultaneously be used for I/O when right justified I/O is desired. Simultaneous addressing and I/O can therefore be carried out.

FIG 4 shows how the microprocessor **50** is able to achieve performance equal to the use of static RAMS with DRAMs through multiple instruction fetch in a single clock cycle and instruction fetch-ahead. Instruction register **108** receives four 8-bit byte instruction words 1–4 on 32-bit internal data bus **90**. The four instruction byte **1–4** locations of the instruction register **108** are connected to multiplexer **170** by busses **172**, **174**, **176** and **178**, respectively. A microprogram counter **180** is connected to the multiplexer **170** by lines **182**. The multiplexer **170** is connected to decoder **184** by bus **186**. The decoder **184** provides internal signals to the rest of the microprocessor **50** on lines **188**.

Most significant bits **190** of each instruction byte **1–4** location are connected to a 4-input decoder **192** by lines **194**. The output of decoder **192** is connected to memory controller **118** by line **196**. Program counter **130** is connected to memory controller **118** by internal address bus **136**, and the instruction register **108** is connected to the memory controller **118** by the internal data bus **90**. Address/data bus **198** and control bus **200** are connected to the DRAMS **150** (FIG 3).

In operation, when the most significant bits **190** of remaining instructions 1–4 are "1" in a clock cycle of the microprocessor **50**, there are no memory reference instructions in the queue. The output of decoder **192** on line **196** requests an instruction fetch ahead by memory controller **118** without interference with other accesses. While the current instructions in instruction register **108** are executing, the memory controller **118** obtains the address of the next set of four instructions from program counter **130** and obtains that set of instructions. By the time the current set of instructions has completed execution, the next set of instructions is ready for loading into the instruction register.

Details of the DMA CPU **72** are provided in FIG 5. Internal data bus **90** is connected to memory controller **118** and to DMA instruction register **210**. The DMA instruction register **210** is connected to DMA program counter **212** by bus **214**, to transfer size counter **216** by bus **218** and to timed transfer interval counter **220** by bus **222**. The DMA instruction register **210** is also connected to DMA I/O and RAM address register **224** by line **226**. The DMA I/O and RAM address register **224** is connected to the memory controller **118** by memory cycle request line **228** and bus **230**. The DMA program counter **212** is connected to the internal address bus **136** by bus **232**. The transfer size counter **216** is connected to a DMA instruction done decrementer **234** by lines **236** and **238**. The decrementer **234** receives a control input on memory cycle acknowledge line **240**. When transfer size counter **216** has completed its count, it provides a control signal to DMA program counter **212** on line **242**. Timed transfer interval counter **220** is connected to decrementer **244** by lines **246** and **248**. The decrementer **244** receives a control input from a microprocessor system clock on line **250**.

The DMA CPU **72** controls itself and has the ability to fetch and execute instructions. It operates as a co-processor to the main CPU **70** (FIG 2) for time specific processing.

FIG 6 shows how the microprocessor **50** is connected to an electrically programmable read only memory (EPROM) **260** by reconfiguring the data lines **52** so that some of the data lines **52** are input lines and some of them are output lines. Data lines **52** D0–D7 provide data to and from corresponding data terminals **262** of the EPROM **260**. Data lines **52** D9–D18 provide addresses to address terminals **264** of the EPROM **260**. Data lines **52** D19–D31 provide inputs from the microprocessor **50** to memory and I/O decode logic **266**. RAS 0:1 control line **268** provides a control signal for determining whether the memory and I/O decode logic provides a DRAM RAS output on line **270** or a column enable output for the EPROM **260** on line **272**. Column address strobe terminal **60** of the microprocessor **50** provides an output enable signal on line **274** to the corresponding terminal **276** of the EPROM **260**.

FIGS. 7 and 8 show the front and back of a one card data processing system **280** incorporating the microprocessor **50**, MSM514258-10 type DRAMs **150** totalling 2 megabytes, a Motorola 50 MegaHertz crystal oscillator clock **282**, I/O circuits **284** and a 27256 type EPROM **260**. The I/O circuits **284** include a 74HC04 type high speed hex inverter circuit **286**, an IDT39C828 type 10-bit inverting buffer circuit **288**, an IDT39C822 type 10-bit inverting register circuit **290**, and two IDT39C823 type 9-bit non-inverting register circuits **292**. The card **280** is completed with a MAX12V type DC—DC converter circuit **294**, 34-pin dual AMP type headers **296**, a coaxial female power connector **298**, and a 3-pin AMP right angle header **300**. The card **280** is a low cost, imbeddable product that can be incorporated in larger systems or used as an internal development tool.

The microprocessor **50** is a very high performance (50 MHz) RISC influenced 32-bit CPU designed to work closely with dynamic RAM. Clock for clock, the microprocessor **50** approaches the theoretical performance limits possible with a single CPU configuration. Eventually, the microprocessor **50** and any other processor is limited by the bus bandwidth and the number of bus paths. The critical conduit is between the CPU and memory.

One solution to the bus bandwidth/bus path problem is to integrate a CPU directly onto the memory chips, giving every memory a direct bus to the CPU. FIG 9 shows another microprocessor **310** that is provided integrally with 1 megabit of DRAM **311** in a single integrated circuit **312**. Until the present invention, this solution has not been practical, because most high performance CPUs require from 500,000 to 1,000,000 transistors and enormous die sizes just by themselves. The microprocessor **310** is equivalent to the microprocessor **50** in FIGS 1–8. The microprocessors **50** and **310** are the most transistor efficient high performance CPUs in existence, requiring fewer than 50,000 transistors for dual processors **70** and **72** (FIG 2) or **314** and **316** (less memory). The very high speed of the microprocessors **50** and **310** is to a certain extent a function of the small number of active devices. In essence, the less silicon gets in the way, the faster the electrons can get where they are going.

The microprocessor **310** is therefore the only CPU suitable for integration on the memory chip die **312**. Some

simple modifications to the basic microprocessor **50** to take advantage of the proximity to the DRAM array **311** can also increase the microprocessor **50** clock speed by 50 percent and probably more

The microprocessor **310** core on board the DRAM die **312** provides most of the speed and functionality required for a large group of applications from automotive to peripheral control However, the integrated CPU **310**/DRAM **311** concept has the potential to redefine significantly the way multiprocessor solutions can solve a spectrum of very compute intensive problems The CPU **310**/DRAM**311** combination eliminates the Von Neumann bottleneck by distributing it across numerous CPU/DRAM chips **312** The microprocessor **310** is a particularly good core for multiprocessing, since it was designed with the SDI targeting array in mind, and provisions were made for efficient interprocessor communications

Traditional multiprocessor implementations have been very expensive in addition to being unable to exploit fully the available CPU horsepower Multiprocessor systems have typically been built up from numerous board level or box level computers The result is usually an immense amount of hardware with corresponding, wiring, power consumption and communications problems By the time the systems are interconnected, as much as 50 percent of the bus speed has been utilized just getting through the interfaces.

In addition, multiprocessor system software has been scarce A multiprocessor system can easily be crippled by an inadequate load-sharing algorithm in the system software, which allows one CPU to do a great deal of work and the others to be idle Great strides have been made recently in systems software, and even UNIX V4 may be enhanced to support multiprocessing. Several commercial products from such manufacturers as DUAL Systems and UNISOFT do a credible job on 68030 type microprocessor systems now.

The microprocessor **310** architecture eliminates most of the interface friction, since up to **64** CPU **310** RAM **311** processors should be able to intercommunicate without buffers or latches Each chip **312** has about 40 MIPS raw speed, because placing the DRAM **311** next to the CPU **310** allows the microprocessor **310** instruction cycle to be cut in half compared to the microprocessor **50** A **64** chip array of these chips **312** is more powerful than any other existing computer Such an array fits on a 3×5 card, cost less than a FAX machine and draw about the same power as a small television.

Dramatic changes in price/performance always reshape existing applications and almost always create new ones The introduction of microprocessors in the mid 1970s created video games, personal computers, automotive computers, electronically controlled appliances, and low cost computer peripherals

The integrated circuit **312** will find applications in all of the above areas, plus create some new ones A common generic parallel processing algorithm handles convolution/Fast Fourier Transform (FFT)/pattern recognition Interesting product possibilities using the integrated circuit **312** include high speed reading machines, real-time speech recognition, spoken language translation, real-time robot vision a product to identify people by their faces, and an automotive or aviation collision avoidance system

A real time processor for enhancing high density television (HDTV) images, or compressing the HDTV information into a smaller bandwidth, would be very feasible The load sharing in HDTV could be very straightforward Splitting up the task according to color and frame would require 6, 9 or 12 processors Practical implementation might require 4 meg RAMs integrated with the microprocessor **310**

The microprocessor **310** has the following specifications:

CONTROL LINES
4—POWER/GROUND
1—CLOCK
32—DATA I/O
4—SYSTEM CONTROL
EXTERNAL MEMORY FETCH
EXTERNAL MEMORY FETCH AUTOINCREMENT X
EXTERNAL MEMORY FETCH AUTOINCREMENT Y
EXTERNAL MEMORY WRITE
EXTERNAL MEMORY WRITE AUTOINCREMENT X
EXTERNAL MEMORY WRITE AUTOINCREMENT Y
EXTERNAL PROM FETCH
LOAD ALL X REGISTERS
LOAD ALL Y REGISTERS
LOAD ALL PC REGISTERS
EXCHANGE X AND Y
INSTRUCTION FETCH
ADD TO PC
ADD TO X
WRITE MAPPING REGISTER
READ MAPPING REGISTER
REGISTER CONFIGURATION
MICROPROCESSOR **310** CPU **316** CORE
COLUMN LATCH1 (1024 BITS) 32×32 MUX
STACK POINTER (16 BITS)
COLUMN LATCH2 (1024 BITS) 32×32 MUX
RSTACK POINTER (16 BITS)
PROGRAM COUNTER 32 BITS
X0 REGISTER 32 BITS (ACTIVATED ONLY FOR ON-CHIP ACCESSES)
Y0 REGISTER 32 BITS (ACTIVATED ONLY FOR ON-CHIP ACCESSES)
LOOP COUNTER 32 BITS
DMA CPU **314** CORE
DMA PROGRAM COUNTER 24 BITS
INSTRUCTION REGISTER 32 BITS
I/O & RAM ADDRESS REGISTER 32 BITS
TRANSFER SIZE COUNTER 12 BITS
INTERVAL COUNTER 12 BITS

To offer memory expansion for the basic chip **312**, an intelligent DRAM can be produced This chip will be optimized for high speed operation with the integrated circuit **312** by having three on-chip address registers: Program Counter, X Register and Y register As a result, to access the intelligent DRAM, no address is required and a total access cycle could be as short as 10 nsec Each expansion DRAM would maintain its own copy of the three registers and would be identified by a code specifying its memory address Incrementing and adding to the three registers will actually take place on the memory chips A maximum of 64 intelligent DRAM peripherals would allow a large system to be created without sacrificing speed by introducing multiplexers or buffers

There are certain differences between the microprocessor **310** and the microprocessor **50** that arise from providing the microprocessor **310** on the same die **312** with the DRAM **311** Integrating the DRAM **311** allows architectural changes in the microprocessor **310** logic to take advantage of existing on-chip DRAM **311** circuitry. Row and column design is

inherent in memory architecture. The DRAMs **311** access random bits in a memory array by first selecting a row of 1024 bits, storing them into a column latch, and then selecting one of the bits as the data to be read or written.

The time required to access the data is split between the row access and the column access. Selecting data already stored in a column latch is faster than selecting a random bit by at least a factor of six. The microprocessor **310** takes advantage of this high speed by creating a number of column latches and using them as caches and shift registers. Selecting a new row of information may be thought of as performing a 1024-bit read or write with the resulting immense bus bandwidth.

1. The microprocessor **50** treats its 32-bit instruction register **108** (see FIGS. **2** and **4**) as a cache for four 8-bit instructions. Since the DRAM **311** maintains a 1024-bit latch for the column bits, the microprocessor **310** treats the column latch as a cache for 128 8-bit instructions. Therefore, the next instruction will almost always be already present in the cache. Long loops within the cache are also possible and more useful than the 4 instruction loops in the microprocessor **50**.

2. The microprocessor **50** uses two 16×32-bit deep register arrays **74** and **134** (FIG. **2**) for the parameter stack and the return stack. The microprocessor **310** creates two other 1024-bit column latches to provide the equivalent of two 32×32-bit arrays, which can be accessed twice as fast as a register array.

3. The microprocessor **50** has a DMA capability which can be used for I/O to a video shift register. The microprocessor **310** uses yet another 1024-bit column latch as a long video shift register to drive a CRT display directly. For color displays, three on-chip shift registers could also be used. These shift registers can transfer pixels at a maximum of 100 MHz.

4. The microprocessor **50** accesses memory via an external 32-bit bus. Most of the memory **311** for the microprocessor **310** is on the same die **312**. External access to more memory is made using an 8-bit bus. The result is a smaller die, smaller package and lower power consumption than the microprocessor **50**.

5. The microprocessor **50** consumes about a third of its operating power charging and discharging the I/O pins and associated capacitances. The DRAMs **150** (FIG. **8**) connected to the microprocessor **50** dissipate most of their power in the I/O drivers. A microprocessor **310** system will consume about one-tenth the power of a microprocessor **50** system, since having the DRAM **311** next to the processor **310** eliminates most of the external capacitances to be charged and discharged.

6. Multiprocessing means splitting a computing task between numerous processors in order to speed up the solution. The popularity of multiprocessing is limited by the expense of current individual processors as well as the limited interprocessor communications ability. The microprocessor **310** is an excellent multiprocessor candidate, since the chip **312** is a monolithic computer complete with memory, rendering it low-cost and physically compact.

The shift registers implemented with the microprocessor **310** to perform video output can also be configured as interprocessor communication links. The INMOS transputer attempted a similar strategy, but at much lower speed and without the performance benefits inherent in the microprocessor **310** column latch architecture. Serial I/O is a prerequisite for many multiprocessor topologies because of the many neighbor processors which communicate. A cube has 6 neighbors. Each neighbor communicates using these lines:

DATA IN

CLOCK IN

READY FOR DATA

DATA OUT

DATA READY

CLOCK OUT

A special start up sequence is used to initialize the on-chip DRAM **311** in each of the processors.

The microprocessor **310** column latch architecture allows neighbor processors to deliver information directly to internal registers or even instruction caches of other chips **312**. This technique is not used with existing processors, because it only improves performance in a tightly coupled DRAM system.

7. The microprocessor **50** architecture offers two types of looping structures: LOOP-IF-DONE and MICRO-LOOP. The former takes an 8-bit to 24-bit operand to describe the entry point to the loop address. The latter performs a loop entirely within the 4 instruction queue and the loop entry point is implied as the first instruction in the queue. Loops entirely within the queue run without external instruction fetches and execute up to three times as fast as the long loop construct. The microprocessor **310** retains both constructs with a few differences. The microprocessor **310** microloop functions in the same fashion as the microprocessor **50** operation, except the queue is 1024-bits or 128 8-bit instructions long. The microprocessor **310** microloop can therefore contain jumps, branches, calls and immediate operations not possible in the 4 8-bit instruction microprocessor **50** queue.

Microloops in the microprocessor **50** can only perform simple block move and compare functions. The larger microprocessor **310** queue allows entire digital signal processing or floating point algorithms to loop at high speed in the queue.

The microprocessor **50** offers four instructions to redirect execution:

CALL

BRANCH

BRANCH-IF-ZERO

LOOP-IF-NOT-DONE

These instructions take a variable length address operand 8, 16 or 24 bits long. The microprocessor **50** next address logic treats the three operands similarly by adding or subtracting them to the current program counter. For the microprocessor **310** the 16 and 24-bit operands function in the same manner as the 16 and 24-bit operands in the microprocessor **50**. The 8-bit class operands are reserved to operate entirely within the instruction queue. Next address decisions can therefore be made quickly, because only 10 bits of addresses are affected, rather than 32. There is no carry or borrow generated past the 10 bits.

8. The microprocessor **310** CPU **316** resides on an already crowded DRAM die **312**. To keep chip size as small as possible, the DMA processor **72** of the microprocessor **50** has been replaced with a more traditional DMA controller **314**. DMA is used with the microprocessor **310** to perform the following functions:

Video output to a CRT

Multiprocessor serial communications

8-bit parallel I/O

The DMA controller **314** can maintain both serial and parallel transfers simultaneously. The following DMA sources and destinations are supported by the microprocessor **310**:

DESCRIPTION	I/O	LINES
1 Video shift register	OUTPUT	1 to 3
2 Multiprocessor serial	BOTH	6 lines channel
3 8-bit parallel	BOTH	8 data, 4 control

The three sources use separate 1024-bit buffers and separate I O pins Therefore, all three may be active simultaneously without interference

The microprocessor **310** can be implemented with either a single multiprocessor serial buffer or separate receive and sending buffers for each channel, allowing simultaneous bidirectional communications with six neighbors simultaneously

FIGS. **10** and **11** provide details of the PROM DMA used in the microprocessor **50**. The microprocessor **50** executes faster than all but the fastest PROMs PROMS are used in a microprocessor **50** system to store program segments and perhaps entire programs The microprocessor **50** provides a feature on power-up to allow programs to be loaded from low-cost, slow speed PROMs into high speed DRAM for execution The logic which performs this function is part of the DMA memory controller **118** The operation is similar to DMA, but not identical since four 8-bit bytes must be assembled on the microprocessor **50** chip then written to the DRAM **150**

The microprocessor **50** directly interfaces to DRAM **150** over a triple multiplexed data and address bus **350**, which carries RAS addresses CAS addresses and data The EPROM **260**, on the other hand, is read with non-multiplexed busses The microprocessor **50** therefore has a special mode which unmultiplexes the data and address lines to read 8 bits of EPROM data Four 8-bit bytes are read in this fashion The multiplexed bus **350** is turned back on, and the data is written to the DRAM **150**

When the microprocessor **50** detects a RESET condition, the processor stops the main CPU **70** and forces a mode **0** (PROM LOAD) instruction into the DMA CPU **72** instruction register The DMA instruction directs the memory controller to read the EPROM **260** data at 8 times the normal access time for memory Assuming a 50 MHz microprocessor **50**, this means an access time of 320 nsec The instruction also indicates:

The selection address of the EPROM **260** to be loaded

The number of 32-bit words to transfer,

The DRAM **150** address to transfer into

The sequence of activities to transfer one 32-bit word from EPROM **260** to DRAM **150** are:

1 RAS goes low at **352**, latching the EPROM **260** select information from the high order address bits The EPROM **260** is selected.

2. Twelve address bits (consisting of what is normally DRAM CAS addresses plus two byte select bits are placed on the bus **350** going to the EPROM **260** address pins These signals will remain on the lines until the data from the EPROM **260** has been read into the microprocessor **50** For the first byte, the byte select bits will be binary 00.

3 CAS goes low at **354**, enabling the EPROM **260** data onto the lower 8 bits of the external address/data bus **350**. NOTE: It is important to recognize that, during this part of the cycle, the lower 8 bits of the external data/address bus are functioning as inputs, but the rest of the bus is still acting as outputs.

4 The microprocessor **50** latches these eight least significant bits internally and shifts them 8 bits left to shift them to the next significant byte position

5 Steps **2**, **3** and **4** are repeated with byte address **01**

6 Steps **2**, **3** and **4** are repeated with byte address **10**

7 Steps **2**, **3** and **4** are repeated with byte address **11**.

8 CAS goes high at **356**, taking the EPROM **260** off the data bus.

9. RAS goes high at **358**, indicating the end of the EPROM **260** access

10 RAS goes low at **360**, latching the DRAM select information from the high order address bits At the same time the RAS address bits are latched into the DRAM **150** The DRAM **150** is selected

11 CAS goes low at **362** latching the DRAM **150** CAS addresses

12. The microprocessor **50** places the previously latched EPROM **260** 32-bit data onto the external address/data bus **350** W goes low at **364**, writing the 32 bits into the DRAM **150**.

13 W goes high at **366** CAS goes high at **368** The process continues with the next word

FIG. **12** shows details of the microprocessor **50** memory controller **118** In operation, bus requests stay present until they are serviced. CPU **70** requests are prioritized at **370** in the order of: 1, Parameter Stack; 2, Return Stack; 3. Data Fetch; 4, Instruction Fetch The resulting CPU request signal and a DMA request signal are supplied as bus requests to bus control **372**, which provides a bus grant signal at **374** Internal address bus **136** and a DMA counter **376** provide inputs to a multiplexer **378** Either a row address or a column address are provided as an output to multiplexed address bus **380** as an output from the multiplexer **378**. The multiplexed address bus **380** and the internal data bus **90** provide address and data inputs, respectively, to multiplexer **382** Shift register **384** supplies row address strobe (RAS) **1** and **2** control signals to multiplexer **386** and column address strobe (CAS) **1** and **2** control signals to multiplexer **388** on lines **390** and **392** The shift register **384** also supplies output enable (OE) and write (W) signals on lines **394** and **396** and a control signal on line **398** to multiplexer **382** The shift register **384** receives a RUN signal on line **400** to generate a memory cycle and supplies a MEMORY READY signal on line **402** when an access is complete

STACK/REGISTER ARCHITECTURE

Most microprocessors use on-chip registers for temporary storage of variables The on-chip registers access data faster than off-chip RAM A few microprocessors use an on-chip push down stack for temporary storage

A stack has the advantage of faster operation compared to on-chip registers by avoiding the necessity to select source and destination registers (A math or logic operation always uses the top two stack items as source and the top of stack as destination) The stack's disadvantage is that it makes some operations clumsy. Some compiler activities in particular require on-chip registers for efficiency

As shown in FIG **13**, the microprocessor **50** provides both on-chip registers **134** and a stack **74** and reaps the benefits of both.

BENEFITS:

1 Stack math and logic is twice as fast as those available on an equivalent register only machine. Most programmers and optimizing compilers can take advantage of this feature

2 Sixteen registers are available for on-chip storage of local variables which can transfer to the stack for computation The accessing of variables is three to four times as fast as available on a strictly stack machine

The combined stack **74**/register **134** architecture has not been used previously due to inadequate understanding by computer designers of optimizing compilers and the mix of transfer versus math/logic instructions

ADAPTIVE MEMORY CONTROLLER

A microprocessor must be designed to work with small or large memory configurations. As more memory loads are added to the data, address, and control lines, the switching speed of the signals slows down. The microprocessor **50** multiplexes the address/data bus three ways, so timing between the phases is critical. A traditional approach to the problem allocates a wide margin of time between bus phases so that systems will work with small or large numbers of memory chips connected. A speed compromise of as much as 50% is required.

As shown in FIG. **14**, the microprocessor **50** uses a feedback technique to allow the processor to adjust memory bus timing to be fast with small loads and slower with large ones. The OUTPUT ENABLE (OE) line **152** from the microprocessor **50** is connected to all memories **150** on the circuit board. The loading on the output enable line **152** to the microprocessor **50** is directly related to the number of memories **150** connected. By monitoring how rapidly OE **152** goes high after a read, the microprocessor **50** is able to determine when the data hold time has been satisfied and place the next address on the bus.

The level of the OE line **152** is monitored by CMOS input buffer **410** which generates an internal READY signal on line **412** to the microprocessor's memory controller. Curves **414** and **416** of the FIG. **15** graph show the difference in rise time likely to be encountered from a lightly to heavily loaded memory system. When the OE line **152** has reached a predetermined level to generate the READY signal, driver **418** generates an OUTPUT ENABLE signal on OE line **152**.

SKIP WITHIN THE INSTRUCTION CACHE

The microprocessor **50** fetches four 8-bit instructions each memory cycle and stores them in a 32-bit instruction register **108**, as shown in FIG. **16**. A class of "test and skip" instructions can very rapidly execute a very fast jump operation within the four instruction cache.

SKIP CONDITIONS:

Always
ACC non-zero
ACC negative
Carry flag equal logic one
Never
ACC equal zero
ACC positive
Carry flag equal logic zero

The SKIP instruction can be located in any of the four byte positions **420** in the 32-bit instruction register **108**. If the test is successful, SKIP will jump over the remaining one, two, or three 8-bit instructions in the instruction register **108** and cause the next four-instruction group to be loaded into the register **108**.

As shown, the SKIP operation is implemented by resetting the 2-bit microinstruction counter **180** to zero on line **422** and simultaneously latching the next instruction group into the register **108**. Any instructions following the SKIP in the instruction register are overwritten by the new instructions and not executed.

The advantage of SKIP is that optimizing compilers and smart programmers can often use it in place of the longer conditional JUMP instruction. SKIP also makes possible microloops which exit when the loop counts down or when the SKIP jumps to the next instruction group. The result is very fast code.

Other machines (such as the PDP-8 and Data General NOVA) provide the ability to skip a single instruction. The microprocessor **50** provides the ability to skip up to three instructions.

MICROLOOP IN THE INSTRUCTION CACHE

The microprocessor **50** provides the MICROLOOP instruction to execute repetitively from one to three instructions residing in the instruction register **108**. The microloop instruction works in conjunction with the LOOP COUNTER **92** (FIG. 2) connected to the internal data bus **90**. To execute a microloop, the program stores a count in LOOP COUNTER **92**. MICROLOOP may be placed in the first, second, third, or last byte **420** of the instruction register **108**. If placed in the first position, execution will just create a delay equal to the number stored in LOOP COUNTER **92** times the machine cycle. If placed in the second, third, or last byte **420**, when the microloop instruction is executed it will test the LOOP COUNT for zero. If zero, execution will continue with the next instruction. If not zero, the LOOP COUNTER **92** is decremented and the 2-bit microinstruction counter is cleared, causing the preceding instructions in the instruction register to be executed again.

Microloop is useful for block move and search operations. By executing a block move completely out of the instruction register **108**, the speed of the move is doubled, since all memory cycles are used by the move rather than being shared with instruction fetching. Such a hardware implementation of microloops is much faster than conventional software implementation of a comparable function.

OPTIMAL CPU CLOCK SCHEME

The designer of a high speed microprocessor must produce a product which operate over wide temperature ranges, wide voltage swings, and wide variations in semiconductor processing. Temperature, voltage, and process all affect transistor propagation delays. Traditional CPU designs are done so that with the worse case of the three parameters, the circuit will function at the rated clock speed. The result are designs that must be clocked a factor of two slower than their maximum theoretical performance, so they will operate properly in worse case conditions.

The microprocessor **50** uses the technique shown in FIGS. **17–19** to generate the system clock and its required phases. Clock circuit **430** is the familiar "ring oscillator" used to test process performance. The clock is fabricated on the same silicon chip as the rest of the microprocessor **50**.

The ring oscillator frequency is determined by the parameters of temperature, voltage, and process. At room temperature, the frequency will be in the neighborhood of 100 MHZ. At 70 degrees Centigrade, the speed will be 50 MHZ. The ring oscillator **430** is useful as a system clock, with its stages **431** producing phase **0**-phase **3** outputs **433** shown in FIG. **19**, because its performance tracks the parameters which similarly affect all other transistors on the same silicon die. By deriving system timing from the ring oscillator **430**, CPU **70** will always execute at the maximum frequency possible, but never too fast. For example, if the processing of a particular die is not good resulting in slow transistors, the latches and gates on the microprocessor **50** will operate slower than normal. Since the microprocessor **50** ring oscillator clock **430** is made from the same transistors on the same die as the latches and gates, it too will operate slower (oscillating at a lower frequency), providing compensation which allows the rest of the chip's logic to operate properly.

ASYNCHRONOUS/SYNCHRONOUS CPU

Most microprocessors derive all system timing from a single clock. The disadvantage is that different parts of the system can slow all operations. The microprocessor **50** provides a dual-clock scheme as shown in FIG. **17**, with the CPU **70** operating asynchronously to I/O interface **432** forming part of memory controller **118** (FIG. 2) and the I/O

interface **432** operating synchronously with the external world of memory and I/O devices. The CPU **70** executes at the fastest speed possible using the adaptive ring counter clock **430**. Speed may vary by a factor of four depending upon temperature, voltage, and process. The external world must be synchronized to the microprocessor **50** for operations such as video display updating and disc drive reading and writing. This synchronization is performed by the I/O interface **432**, speed of which is controlled by a conventional crystal clock **434**. The interface **432** processes requests for memory accesses from the microprocessor **50** and acknowledges the presence of I/O data. The microprocessor **50** fetches up to four instructions in a single memory cycle and can perform much useful work before requiring another memory access. By decoupling the variable speed of the CPU **70** from the fixed speed of the I/O interface **432**, optimum performance can be achieved by each. Recoupling between the CPU **70** and the interface **432** is accomplished with handshake signals on lines **436**, with data/addresses passing on bus **90**, **136**.

ASYNCHRONOUS/SYNCHRONOUS CPU IMBEDDED ON A DRAM CHIP

System performance is enhanced even more when the DRAM **311** and CPU **314** (FIG. **9**) are located on the same die. The proximity of the transistors means that DRAM **311** and CPU **314** parameters will closely follow each other. At room temperature, not only would the CPU **314** execute at 100 MHZ., but the DRAM **311** would access fast enough to keep up. The synchronization performed by the I/O interface **432** would be for DMA and reading and writing I/O ports. In some systems (such as calculators) no I/O synchronization at all would be required, and the I/O clock would be tied to the ring counter clock.

VARIABLE WIDTH OPERANDS

Many microprocessors provide variable width operands. The microprocessor **50** handles operands of 8, 16, or 24 bits using the same op-code. FIG. **20** shows the 32-bit instruction register **108** and the 2-bit microinstruction register **180** which selects the 8-bit instruction. Two classes of microprocessor **50** instructions can be greater than 8-bits. JUMP class and IMMEDIATE. A JUMP or IMMEDIATE op-code is 8-bits, but the operand can be 8, 16, or 24 bits long. This magic is possible because operands must be right justified in the instruction register. This means that the least significant bit of the operand is always located in the least significant bit of the instruction register. The microinstruction counter **180** selects which 8-bit instruction to execute. If a JUMP or IMMEDIATE instruction is decoded, the state of the 2-bit microinstruction counter selects the required 8, 16, or 24 bit operand onto the address or data bus. The unselected 8-bit bytes are loaded with zeros by operation of decoder **440** and gates **442**. The advantage of this technique is the saving of a number of op-codes required to specify the different operand sizes in other microprocessors.

TRIPLE STACK CACHE

Computer performance is directly related to the system memory bandwidth. The faster the memories, the faster the computer. Fast memories are expensive, so techniques have been developed to move a small amount of high-speed memory around to the memory addresses where it is needed. A large amount of slow memory is constantly updated by the fast memory, giving the appearance of a large fast memory array. A common implementation of the technique is known as a high-speed memory cache. The cache may be thought of as fast acting shock absorber smoothing out the bumps in memory access. When more memory is required than the shock can absorb, it bottoms out and slow speed memory is accessed. Most memory operations can be handled by the shock absorber itself. The microprocessor **50** architecture has the ALU **80** (FIG. 2) directly coupled to the top two stack locations **76** and **78**. The access time of the stack **74** therefore directly affects the execution speed of the processor. The microprocessor **50** stack architecture is particularly suitable to a triple cache technique, shown in FIG. **21** which offers the appearance of a large stack memory operating at the speed of on-chip latches **450**. Latches **450** are the fastest form of memory device built on the chip, delivering data in as little as 3 nsec. However latches **450** require large numbers of transistors to construct. On-chip RAM **452** requires fewer transistors than latches, but is slower by a factor of five (15 nsec access). Off-chip RAM **150** is the slowest storage of all. The microprocessor **50** organizes the stack memory hierarchy as three interconnected stacks **450**, **452** and **454**. The latch stack **450** is the fastest and most frequently used. The on-chip RAM stack **452** is next. The off-chip RAM stack **454** is slowest. The stack modulation determines the effective access time of the stack. If a group of stack operations never push or pull more than four consecutive items on the stack, operations will be entirely performed in the 3 nsec latch stack. When the four latches **456** are filled, the data in the bottom of the latch stack **450** is written to the top of the on-chip RAM stack **452**. When the sixteen locations **458** in the on-chip RAM stack **452** are filled, the data in the bottom of the on-chip RAM stack **452** is written to the top of the off-chip RAM stack **454**. When popping data off a full stack **450**, four pops will be performed before stack empty line **460** from the latch stack pointer **462** transfers data from the on-chip RAM stack **452**. By waiting for the latch stack **450** to empty before performing the slower on-chip RAM access, the high effective speed of the latches **456** are made available to the processor. The same approach is employed with the on-chip RAM stack **452** and the off-chip RAM stack **454**.

POLYNOMIAL GENERATION INSTRUCTION

Polynomials are useful for error correction, encryption, data compression, and fractal generation. A polynomial is generated by a sequence of shift and exclusive OR operations. Special chips are provided for this purpose in the prior art.

The microprocessor **50** is able to generate polynomials at high speed without external hardware by slightly modifying how the ALU **80** works. As shown in FIG. **22**, a polynomial is generated by loading the "order" (also known as the feedback terms) into C Register **470**. The value thirty one (resulting in 32 iterations) is loaded into DOWN COUNTER **472**. A register **474** is loaded with zero. B register **476** is loaded with the starting polynomial value. When the POLY instruction executes, C register **470** is exclusively ORed with A register **474** if the least significant bit of B register **476** is a one. Otherwise, the contents of the A register **474** passes through the ALU **80** unaltered. The combination of A and B is then shifted right (divided by 2) with shifters **478** and **480**. The operation automatically repeats the specified number of iterations, and the resulting polynomial is left in A register **474**.

FAST MULTIPLY

Most microprocessors offer a 16×16 or 32×32 bit multiply instruction. Multiply when performed sequentially takes one shift/add per bit, or 32 cycles for 32 bit data. The microprocessor **50** provides a high speed multiply which allows multiplication by small numbers using only a small number of cycles. FIG. **23** shows the logic used to implement the high speed algorithm. To perform a multiply, the size of the multiplier less one is placed in the DOWN COUNTER **472**.

For a four bit multiplier, the number three would be stored in the DOWN COUNTER **472**. Zero is loaded into the A register **474**. The multiplier is written bit reversed into the B Register **476**. For example, a bit reversed five (binary 0101) would be written into B as 1010. The multiplicand is written into the C register **470**. Executing the FAST MULT instruction will leave the result in the A Register **474**, when the count has been completed. The fast multiply instruction is important because many applications scale one number by a much smaller number. The difference in speed between multiplying a 32×32 bit and a 32×4 bit is a factor of 8. If the least significant bit of the multiplier is a "ONE" the contents of the A register **474** and the C register **470** are added. If the least significant bit of the multiplier is a "ZERO", the contents of the A register are passed through the ALU **80** unaltered. The output of the ALU **80** is shifted left by shifter **482** in each iteration. The contents of the B register **476** are shifted right by the shifter **480** in each iteration.

INSTRUCTION EXECUTION PHILOSOPHY

The microprocessor **50** uses high speed D latches in most of the speed critical areas. Slower on-chip RAM is used as secondary storage.

The microprocessor **50** philosophy of instruction execution is to create a hierarchy of speed as follows:

Logic and D latch transfers	1 cycle	20 nsec
Math	2 cycles	40 nsec
Fetch/store on-chip RAM	2 cycles	40 nsec
Fetch/store in current RAS page	4 cycles	80 nsec
Fetch/store with RAS cycle	11 cycles	220 nsec

With a 50 MHZ clock, many operations can be performed in 20 nsec and almost everything else in 40 nsec.

To maximize speed certain techniques in processor design have been used. They include:

Eliminating arithmetic operations on addresses,

Fetching up to four instructions per memory cycle.,

Pipelineless instruction decoding

Generating results before they are needed

Use of three level stack caching

PIPELINE PHILOSOPHY

Computer instructions are usually broken down into sequential pieces, for example: fetch, decode., register read, execute., and store. Each piece will require a single machine cycle. In most Reduced Instruction Set Computer (RISC) chips, instruction require from three to six cycles.

RISC instructions are very parallel. For example, each of 70 different instructions in the SPARC (SUN Computer's RISC chip) has five cycles. Using a technique called "pipelining", the different phases of consecutive instructions can be overlapped.

To understand pipelining, think of building five residential homes. Each home will require in sequence, a foundation, framing, plumbing and wiring, roofing and interior finish. Assume that each activity takes one week. To build one house will take five weeks.

But what if you want to build an entire subdivision? You have only one of each work crew, but when the foundation men finish on the first house, you immediately start them on the second one, and so on. At the end of five weeks, the first home is complete., but you also have five foundations. If you have kept the framing, plumbing, roofing, and interior guys all busy, from five weeks on., a new house will be completed each week.

This is the way a RISC chip like SPARC appears to execute an instruction in a single machine cycle. In reality., a RISC chip is executing one fifth of five instructions each machine cycle. And if five instructions stay in sequence, an instruction will be completed each machine cycle.

The problems with a pipeline are keeping the pipe full with instructions. Each time an out of sequence instruction such as a BRANCH or CALL occurs, the pipe must be refilled with the next sequence. The resulting dead time to refill the pipeline can become substantial when many IF/THEN/ELSE statements or subroutines are encountered.

THE PIPELINE APPROACH

The microprocessor **50** has no pipeline as such. The approach of this microprocessor to speed is to overlap instruction fetching with execution of the previously fetched instruction(s). Beyond that, over half the instructions (the most common ones) execute entirely in a single machine cycle of 20 nsec. This is possible because:

1. Instruction decoding resolves in 2.5 nsec.

2. Incremented, decremented and some math values are calculated before they are needed., requiring only a latching signal to execute.

3. Slower memory is hidden from high speed operations by high-speed D latches which access in 4 nsec. The disadvantage for this microprocessor is a more complex chip design process. The advantage for the chip user is faster ultimate throughput since pipeline stalls cannot exist. Pipeline synchronization with availability flag bits and other such pipeline handling is not required by this microprocessor.

For example, in some RISC machines an instruction which tests a status flag may have to wait for up to four cycles for the flag set by the previous instruction to be available to be tested. Hardware and software debugging is also somewhat easier because the user doesn't have to visualize five instructions simultaneously in the pipe.

OVERLAPPING INSTRUCTION FETCH/EXECUTE

The slowest procedure the microprocessor **50** performs is to access memory. Memory is accessed when data is read or written. Memory is also read when instructions are fetched. The microprocessor **50** is able to hide fetch of the next instruction behind the execution of the previously fetched instruction(s). The microprocessor **50** fetches instructions in 4-byte instruction groups. An instruction group may contain from one to four instructions. The amount of time required to execute the instruction group ranges from 4 cycles for simple instructions to 64 cycles for a multiply.

When a new instruction group is fetched, the microprocessor instruction decoder looks at the most significant bit of all four of the bytes. The most significant bit of an instruction determines if a memory access is required. For example CALL., FETCH, and STORE all require a memory access to execute. If all four bytes have nonzero most significant bits, the microprocessor initiates the memory fetch of the next sequential 4-byte instruction group. When the last instruction in the group finishes executing, the next 4-byte instruction group is ready and waiting on the data bus needing only to be latched into the instruction register. If the 4-byte instruction group required four or more cycles to execute and the next sequential access was a column address strobe (CAS) cycle., the instruction fetch was completely overlapped with execution.

INTERNAL ARCHITECTURE

The microprocessor **50** architecture consists of the following:

PARAMETER STACK <--> ALU* <--> RETURN STACK <--> Y REGISTER	
<---32 BITS---> 16 DEEP	<---32 BITS---> 16 DEEP
Used for math and logic	Used for subroutine and interrupt return addresses as well as local variables
Push down stack	Push down stack
Can overflow into off-chip RAM	Can overflow into off-chip RAM
	Can also be accessed relative to top of stack
LOOP COUNTER	(32-bits, can decrement by 1) Used by class of test and loop instructions
X REGISTER	(32-bits, can increment or decrement by 4) Used to point to RAM locations.
PROGRAM COUNTER	(32-bits, increments by 4) Points to 4-byte instruction groups in RAM
INSTRUCTION REG	(32-Bits) Holds 4-byte instruction groups while they are being decoded and executed

*Math and logic operations use the TOP item and NEXT to top Parameter Stack items as the operands. The result is pushed onto the Parameter Stack.

*Return addresses from subroutines are placed on the Return Stack. The Y REGISTER is used as a pointer to RAM locations. Since the Y REGISTER is the top item of the Return Stack, nesting of indices is straightforward.

MODE—A register with mode and status bits.

MODE-BITS:

—Slow down memory accesses by 8 if "1". Run full speed if "0". (Provided for access to slow EPROM.)

—Divide the system clock by 1023 if "1" to reduce power consumption. Run full speed if "0". (On-chip counters slow down if this bit is set.)

—Enable external interrupt 1.

—Enable external interrupt 2.

—Enable external interrupt 3.

—Enable external interrupt 4.

—Enable external interrupt 5.

—Enable external interrupt 6.

—Enable external interrupt 7.

ON-CHIP MEMORY LOCATIONS:

MODE-BITS.

DMA-POINTER.

DMA-COUNTER.

STACK-POINTER—Pointer into Parameter Stack.

STACK-DEPTH—Depth of on-chip Parameter Stack.

RSTACK-POINTER—Pointer into Return Stack.

RSTACK-DEPTH—Depth of on-chip Return Stack.

ADDRESSING MODE HIGH POINTS

The data bus is 32-bits wide. All memory fetches and stores are 32-bits. Memory bus addresses are 30 bits. The least significant 2 bits are used to select one-of-four bytes in some addressing modes. The Program Counter, X Register, and Y Register are implemented as D latches with their outputs going to the memory address bus and the bus incrementer/decrementer. Incrementing one of these registers can happen quickly, because the incremented value has already rippled through the inc/dec logic and need only be clocked into the latch. Branches and Calls are made to 32-bit word boundaries.

INSTRUCTION SET

32-BIT INSTRUCTION FORMAT

The thirty two bit instructions are CALL, BRANCH, BRANCH-IF-ZERO, and LOOP-IF-NOT-DONE. These instructions require the calculation of an effective address. In many computers the effective address is calculated by adding or subtracting an operand with the current Program Counter. This math operation requires from four to seven machine cycles to perform and can definitely bog down machine execution. The microprocessor's strategy is to perform the required math operation at assembly or linking time and do a much simpler "Increment to next page" or "Decrement to previous page" operation at run time. As a result, the microprocessor branches execute in a single cycle.

24-BIT OPERAND FORM

Byte 1	Byte 2	Byte 3	Byte 4
WWWWWW XX -	YYYYYYYY -	YYYYYYYY -	YYYYYYYY

With a 24-bit operand, the current page is considered to be defined by the most significant 6 bits of the Program Counter.

16-BIT OPERAND FORM:

QQQQQQQQ - WWWWWWW XX - YYYYYYYY - YYYYYYYY With a 16-bit operand, the current page is considered to be defined by the most significant 14 bits of the Program Counter.

8-BIT OPERAND FORM:

QQQQQQQQ - QQQQQQQQ - WWWWWW XX - YYYYYYYY With an 8-bit operand, the current page is considered to be defined by the most significant 22 bits of the Program Counter.

QQQQQQQQ—Any 8-bit instruction.

WWWWWW—Instruction op-code.

XX—Select how the address bits will be used:

00 —Make all high-order bits zero (Page zero addressing)

01 —Increment the high-order bits. (Use next page)

10 —Decrement the high-order bits. (Use previous page)

11 —Leave the high-order bits unchanged. (Use current page)

YYYYYYYY —The address operand field. This field is always shifted left two bits (to generate a word rather than byte address) and loaded into the Program Counter. The microprocessor instruction decoder figures out the width of the operand field by the location of the instruction op-code in the four bytes.

The compiler or assembler will normally use the shortest operand required to reach the desired address so that the leading bytes can be used to hold other instructions. The effective address is calculated by combining:

The current Program Counter,

The 8, 16, or 24 bit address operand in the instruction,

Using one of the four allowed addressing modes

EXAMPLES OF EFFECTIVE ADDRESS CALCULATION

EXAMPLE 1

Byte 1	Byte 2	Byte 3	Byte 4
QQQQQQQQ	QQQQQQQQ	00000011	10011000

The "QQQQQQQQs" in Byte **1** and **2** indicate space in the 4-byte memory fetch which could be hold two other instructions to be executed prior to the CALL instruction Byte **3** indicates a CALL instruction (six zeros) in the current page (indicated by the 11 bits) Byte 4 indicates that the hexadecimal number **98** will be forced into the Program Counter bits **2** through **10** (Remember, a CALL or BRANCH always goes to a word boundary so the two least significant bits are always set to zero) The effect of this instruction would be to CALL a subroutine at WORD location HEX **98** in the current page. The most significant 22 bits of the Program Counter define the current page and will be unchanged

EXAMPLE 2

Byte 1	Byte 2	Byte 3	Byte 4
000001 01	00000001	00000000	00000000

If we assume that the Program Counter was HEX 0000 0156 which is binary: 00000000 00000000 00000001 01010110 =OLD PROGRAM COUNTER Byte **1** indicates a BRANCH instruction op code (000001) and "01" indicates select the next page. Byte **2**,**3**, and **4** are the address operand These 24-bits will be shifted to the left two places to define a WORD address HEX 0156 shifted left two places is HEX 0558. Since this is a 24-bit operand instruction, the most significant 6 bits of the Program Counter define the current page These six bits will be incremented to select the next page Executing this instruction will cause the Program Counter to be loaded with HEX 0400 0558 which is binary:

00000100 00000000 00000101 01011000 =NEW PROGRAM COUNTER

INSTRUCTIONS

CALL-LONG

0000 00XX - YYYYYYYY - YYYYYYYY - YYYYYYYY

Load the Program Counter with the effective WORD address specified Push the current PC contents onto the RETURN STACK.

OTHER EFFECTS: CARRY or modes no effect May cause Return Stack to force an external memory cycle if on-chip Return Stack is full

BRANCH

0000 01XX - YYYYYYYY - YYYYYYYY - YYYYYYYY

Load the Program Counter with the effective WORD address specified

OTHER EFFECTS: NONE

BRANCH-IF-ZERO

0000 10XX - YYYYYYYY - YYYYYYYY - YYYYYYYY

Test the TOP value on the Parameter Stack. If the value is equal to zero, load the Program Counter with the effective WORD address specified. If the TOP value is not equal to zero, increment the Program Counter and fetch and execute the next instruction

OTHER EFFECTS: NONE

LOOP-IF-NOT-DONE

0000 11 YY - (XXXX XXXX) - (XXXX XXXX) - (XXXX XXXX)

If the LOOP COUNTER is not zero, load the Program Counter with the effective WORD address specified. If the LOOP COUNTER is zero, decrement the LOOP COUNTER, increment the Program Counter and fetch and execute the next instruction.

OTHER EFFECTS: NONE

8-BIT INSTRUCTIONS PHILOSOPHY

Most of the work in the microprocessor **50** is done by the 8-bit instructions Eight bit instructions are possible with the microprocessor because of the extensive use of implied stack addressing Many 32-bit architectures use 8-bits to specify the operation to perform but use an additional 24-bits to specify two sources and a destination

For math and logic operations, the microprocessor **50** exploits the inherent advantage of a stack by designating the source operand(s) as the top stack item and the next stack item The math or logic operation is performed, the operands are popped from the stack, and the result is pushed back on the stack. The result is a very efficient utilization of instruction bits as well as registers A comparable situation exists between Hewlett Packard calculators (which use a stack) and Texas Instrument calculators which don't The identical operation on an HP will require one half to one third the keystrokes of the TI.

The availability of 8-bit instructions also allows another architectural innovation, the fetching of four instructions in a single 32-bit memory cycle The advantages of fetching multiple instructions are:

Increased execution speed even with slow memories,

Similar performance to the Harvard (separate data and instruction busses) without the expense.

Opportunities to optimize groups of instructions,

The capability to perform loops within this mini-cache

The microloops inside the four instruction group are effective for searches and block moves.

SKIP INSTRUCTIONS

The microprocessor **50** fetches instructions in 32-bit chunks called 4-byte instruction groups These four bytes may contain four 8-bit instructions or some mix of 8-bit and 16 or 24-bit instructions. SKIP instructions in the microprocessor skip any remaining instructions in a 4-byte instruction group and cause a memory fetch to get the next 4-byte instruction group Conditional SKIPs when combined with 3-byte BRANCHES will create conditional BRANCHES SKIPs may also be used in situations when no use can be made of the remaining bytes in a 4-instruction group A SKIP executes in a single cycle, whereas a group of three NOPs would take three cycles

SKIP-ALWAYS—Skip any remaining instructions in this 4-byte instruction group

Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group

SKIP-IF-ZERO—If the TOP item of the Parameter Stack is zero, skip any remaining instructions in the 4-byte instruction group Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group

If the TOP item is not zero, execute the next sequential instruction.

SKIP-IF-POSITIVE—If the TOP item of the Parameter Stack has a the most significant bit (the sign bit) equal to "0" skip any remaining instructions in the 4-byte instruction

group Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group If the TOP item is not "0" execute the next sequential instruction

SKIP-IF-NO-CARRY—If the CARRY flag from a SHIFT or arithmetic operation is not equal to "1", skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group If the CARRY is equal to "1", execute the next sequential instruction.

SKIP-NEVER—Execute the next sequential (NOP) instruction. (Delay one machine cycle).

SKIP-IF-NOT-ZERO - If the TOP item on the Parameter Stack is not equal to "0", skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group

If the TOP item is equal 0" execute the next sequential instruction

SKIP-IF-NEGATIVE—If the TOP item on the Parameter Stack has its most significant bit (sign bit) set to "1", skip any remaining instructions in the 4-byte instruction group Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group If the TOP item has its most significant bit set to "0", execute the next sequential instruction

SKIP-IF-CARRY—If the CARRY flag is set to 1 as a result of SHIFT or arithmetic operation, skip any remaining instructions in the 4-byte instruction group Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group If the CARRY flag is "0", execute the next sequential instruction

MICROLOOPS

Microloops are a unique feature of the microprocessor architecture which allows controlled looping within a 4-byte instruction group. A microloop instruction tests the LOOP COUNTER for "0" and may perform an additional test If the LOOP COUNTER is not "0" and the test is met, instruction execution continues with the first instruction in the 4-byte instruction group, and the LOOP COUNTER is decremented A microloop instruction will usually be the last byte in a 4-byte instruction group, but it can be any byte If the LOOP COUNTER is "0" or the test is not met, instruction execution continues with the next instruction If the microloop is the last byte in the 4-byte instruction group, the most significant 30:bits of the Program Counter are incremented and the next 4-byte instruction group is fetched from memory On a termination of the loop on LOOP COUNTER equal to "0", the LOOP COUNTER will remain at "0". Microloops allow short iterative work such as moves and searches to be performed without slowing down to fetch instructions from memory

EXAMPLE

Byte 1 FETCH-VIA-X-AUTOINCREMENT	Byte 2 STORE-VIA-Y-AUTOINCREMENT
Byte 3 ULOOP-UNTIL-DONE	Byte 4 QQQQQQQQ

This example will perform a block move To initiate the transfer. X will be loaded with the starting address of the source Y will be loaded with the starting address of the destination. The LOOP COUNTER will be loaded with the number of 32-bit words to move The microloop will FETCH and STORE and count down the LOOP COUNTER until it reaches zero QQQQQQQQ indicates any instruction can follow.

MICROLOOP INSTRUCTIONS

ULOOP-UNTIL-DONE—If the LOOP COUNTER is not "0", continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER If the LOOP COUNTER is "0", continue execution with the next instruction.

ULOOP-IF-ZERO—If the LOOP COUNTER is not "0" and the TOP item on the Parameter Stack is "0", continue execution with the first instruction in the 4-byte instruction group Decrement the LOOP COUNTER If the LOOP COUNTER is "0" or the TOP item is "1", continue execution with the next instruction

ULOOP-IF-POSITIVE—If the LOOP COUNTER is not "0" and the most significant bit (sign bit) is "0", continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER If the LOOP COUNTER is "0" or the TOP item is "1", continue execution with the next instruction.

ULOOP-IF-NOT-CARRY-CLEAR—If the LOOP COUNTER is not "0" and the floating point exponents found in TOP and NEXT are not aligned, continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER If the LOOP COUNTER is "0" or the exponents are aligned continue execution with the next instruction

This instruction is specifically designed for combination with special SHIFT instructions to align two floating point numbers

ULOOP-NEVER—(DECREMENT-LOOP-COUNTER) Decrement the LOOP COUNTER Continue execution with the next instruction

ULOOP-IF-NOT-ZERO—If the LOOP COUNTER is not "0" and the TOP item of the Parameter Stack is "0", continue execution with the first instruction in the 4-byte instruction group Decrement the LOOP COUNTER If the LOOP COUNTER is "0" or the TOP item is "1", continue execution with the next instruction.

ULOOP-IF-NEGATIVE—If the LOOP COUNTER is not "0" and the most significant bit (sign bit) of the TOP item of the Parameter Stack is "1", continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER If the LOOP COUNTER is "0" or the most significant bit of the Parameter Stack is "0", continue execution with the next instruction

ULOOP-IF-CARRY-SET—If the LOOP COUNTER is not "0" and the exponents of the floating point numbers found in TOP and NEXT are not aligned, continue execution with the first instruction in the 4-byte instruction group Decrement the LOOP COUNTER If the

LOOP COUNTER is "0" or the exponents are aligned, continue execution with the next instruction.

RETURN FROM SUBROUTINE OR INTERRUPT

Subroutine calls and interrupt acknowledgements cause a redirection of normal program execution. In both cases, the current Program Counter is pushed onto the Return Stack so the microprocessor can return to its place in the program after executing the subroutine or interrupt service routine.

NOTE: When a CALL to subroutine or interrupt is acknowledged the Program Counter has already been incremented and is pointing to the 4-byte instruction group following the 4-byte group currently being executed. The instruction decoding logic allows the microprocessor to perform a test and execute a return conditional on the outcome of the test in a single cycle. A RETURN pops an address from the Return Stack and stores it to the Program Counter.

RETURN INSTRUCTIONS

RETURN-ALWAYS—Pop the top item from the Return Stack and transfer it to the Program Counter.

RETURN-IF-ZERO—If the TOP item on the Parameter Stack is "0", pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

RETURN-IF-POSITIVE—If the most significant bit (sign bit) of the TOP item on the Parameter Stack is a "0", pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

RETURN-IF-CARRY-CLEAR—If the exponents of the floating point numbers found in TOP and NEXT are not aligned, pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

RETURN-NEVER—Execute the next instruction (NOP).

RETURN-IF-NOT-ZERO—If the TOP item on the Parameter Stack is not "0", pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

RETURN-IF-NEGATIVE—If the most significant bit (sign bit) of the TOP item on the Parameter Stack is a "1", pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

RETURN-IF-CARRY-SET—If the exponents of the floating point numbers found in TOP and NEXT are aligned, pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

HANDLING MEMORY FROM DYNAMIC RAM

The microprocessor **50**, like any RISC type architecture is optimized to handle as many operations as possible on-chip for maximum speed. External memory operations take from 80 nsec. to 220 nsec. compared with on-chip memory speeds of from 4 nsec. to 30 nsec. There are times when external memory must be accessed.

External memory is accessed using three registers:

X-REGISTER—A 30-bit memory pointer which can be used for memory access and simultaneously incremented or decremented.

Y-REGISTER—A 30-bit memory pointer which can be used for memory access and simultaneously incremented or decremented.

PROGRAM-COUNTER—A 30-bit memory pointer normally used to point to 4-byte instruction groups. External memory may be accessed at addresses relative to the PC. The operands are sometimes called "Immediate" or "Literal" in other computers. When used as memory pointer, the PC is also incremented after each operation.

MEMORY LOAD & STORE INSTRUCTIONS

FETCH-VIA-X—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. X is unchanged.

FETCH-VIA-Y—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. Y is unchanged.

FETCH-VIA-X-AUTOINCREMENT—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. After fetching, increment the most significant 30 bits of X to point to the next 32-bit word address.

FETCH-VIA-Y-AUTOINCREMENT—Fetch the 32-bit memory content pointed to by Y and push it onto the Parameter Stack. After fetching, increment the most significant 30 bits of Y to point to the next 32-bit word address.

FETCH-VIA-X-AUTODECREMENT—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. After fetching, decrement the most significant 30 bits of X to point to the previous 32-bit word address.

FETCH-VIA-Y-AUTODECREMENT—Fetch the 32-bit memory content pointed to by Y and push it onto the Parameter Stack. After fetching, decrement the most significant 30 bits of Y to point to the previous 32-bit word address.

STORE-VIA-X—Pop the top item of the Parameter Stack and store it in the memory location pointed to by X. X is unchanged.

STORE-VIA-Y—Pop the top item of the Parameter Stack and store it in the memory location pointed to by Y. Y is unchanged.

STORE-VIA-X-AUTOINCREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by X. After storing, increment the most significant 30 bits of X to point to the next 32-bit word address.

STORE-VIA-Y-AUTOINCREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by Y. After storing, increment the most significant 30 bits of Y to point to the next 32-bit word address.

STORE-VIA-X-AUTODECREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by X. After storing, decrement the most significant 30 bits of X to point to the previous 32-bit word address.

STORE-VIA-Y-AUTODECREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by Y. After storing, decrement the most significant 30 bits of Y to point to the previous 32-bit word address.

FETCH-VIA-PC—Fetch the 32-bit memory content pointed to by the Program Counter and push it onto the Parameter Stack. After fetching, increment the most significant 30 bits of the Program Counter to point to the next 32-bit word address.

*NOTE: When this instruction executes, the PC is pointing to the memory location following the instruction. The effect

is of loading a 32-bit immediate operand. This is an 8-bit instruction and therefore will be combined with other 8-bit instructions in a 4-byte instruction fetch. It is possible to have from one to four FETCH-VIA-PC instructions in a 4-byte instruction fetch. The PC increments after each execution of FETCH-VIA-PC, so it is possible to push four immediate operands on the stack. The four operands would be the found in the four memory locations following the instruction.

BYTE-FETCH-VIA-X—Fetch the 32-bit memory content pointed to by the most significant 30 bits of X. Using the two least significant bits of X, select one of four bytes from the 32-bit memory fetch, right justify the byte in a 32-bit field and push the selected byte preceded by leading zeros onto the Parameter Stack.

BYTE-STORE-VIA-X—Fetch the 32-bit memory content pointed to by the most significant 30 bits of X. Pop the TOP item from the Parameter Stack. Using the two least significant bits of X place the least significant byte into the 32-bit memory data and write the 32-bit entity back to the location pointed to by the most significant 30 bits of X.

OTHER EFFECTS OF MEMORY ACCESS INSTRUCTIONS:

Any FETCH instruction will push a value on the Parameter Stack **74**. If the on-chip stack is full, the stack will overflow into off-chip memory stack resulting in an additional memory cycle. Any STORE instruction will pop a value from the Parameter Stack **74**. If the on-chip stack is empty, a memory cycle will be generated to fetch a value from off-chip memory stack.

HANDLING ON-CHIP VARIABLES

High-level languages often allow the creation of LOCAL VARIABLES. These variables are used by a particular procedure and discarded. In cases of nested procedures, layers of these variables must be maintained. On-chip storage is up to five times faster than off-chip RAM, so a means of keeping local variables on-chip can make operations run faster. The microprocessor **50** provides the capability for both on-chip storage of local variables and nesting of multiple levels of variables through the Return Stack.

The Return Stack **134** is implemented as 16 on-chip RAM locations. The most common use for the Return Stack **134** is storage of return addresses from subroutines and interrupt calls. The microprocessor allows these 16 locations to also be used as addressable registers. The 16 locations may be read and written by two instructions which indicate a Return Stack relative address from 0–15. When high-level procedures are nested, the current procedure variables push the previous procedure variables further down the Return Stack **134**. Eventually, the Return Stack will automatically overflow into off-chip RAM.

ON-CHIP VARIABLE INSTRUCTIONS

READ-LOCAL-VARIABLE XXXX—Read the XXXXth location relative to the top of the Return Stack (XXXX is a binary number from 0000–1111). Push the item read onto the Parameter Stack. OTHER EFFECTS: If the Parameter Stack is full, the push operation will cause a memory cycle to be generated as one item of the stack is automatically stored to external RAM. The logic which selects the location performs a modulo **16** subtraction. If four local variables have been pushed onto the Return Stack, and an instruction attempts to READ the fifth item, unknown data will be returned.

WRITE-LOCAL-VARIABLE XXXX—Pop the TOP item of the Parameter Stack and write it into the XXXXth location relative to the top of the Return Stack (XXXX is a binary number from 0000–1111). OTHER EFFECTS: If the Parameter Stack is empty, the pop operation will cause a memory cycle to be generated to fetch the Parameter Stack item 30 from external RAM. The logic which selects the location performs a modulo **16** subtraction. If four local variables have been pushed onto the Return Stack, and an instruction attempts to WRITE to the fifth item, it is possible to clobber return addresses or wreak other havoc.

REGISTER AND FLIP-FLOP TRANSFER AND PUSH INSTRUCTIONS

DROP—Pop the TOP item from the Parameter Stack and discard it.

SWAP—Exchange the data in the TOP Parameter Stack location with the data in the NEXT Parameter Stack location.

DUP—Duplicate the TOP item on the Parameter Stack and push it onto the Parameter Stack.

PUSH-LOOP-COUNTER—Push the value in LOOP COUNTER onto the Parameter Stack.

POP-RSTACK-PUSH-TO-STACK—Pop the top item from the Return Stack and push it onto the Parameter Stack.

PUSH-X-REG—Push the value in the X Register onto the Parameter Stack.

PUSH-STACK-POINTER—Push the value of the Parameter Stack pointer onto the Parameter Stack.

PUSH-RSTACK-POINTER—Push the value of the Return Stack pointer onto the Return Stack.

PUSH-MODE-BITS—Push the value of the MODE REGISTER onto the Parameter Stack.

PUSH-INPUT—Read the 10 dedicated input bits and push the value (right justified and padded with leading zeros) onto the Parameter Stack.

SET-LOOP-COUNTER—Pop the TOP value from the Parameter Stack and store it into LOOP COUNTER.

POP-STACK-PUSH-TO-RSTACK—Pop the TOP item from the Parameter Stack and push it onto the Return Stack.

SET-X-REG—Pop the TOP item from the Parameter Stack and store it into the X Register.

SET-STACK-POINTER—Pop the TOP item from the Parameter Stack and store it into the Stack Pointer.

SET-RSTACK-POINTER—Pop the TOP item from the Parameter Stack and store it into the Return Stack Pointer.

SET-MODE-BITS—Pop the TOP value from the Parameter Stack and store it into the MODE BITS.

SET-OUTPUT—Pop the TOP item from the Parameter Stack and output it to the 10 dedicated output bits. OTHER EFFECTS: Instructions which push or pop the Parameter Stack or Return Stack may cause a memory cycle as the stacks overflow back and forth between on-chip and off-chip memory.

LOADING A SHORT LITERAL

A special case of register transfer instruction is used to push an 8-bit literal onto the Parameter Stack. This instruction requires that the 8-bits to be pushed reside in the last byte of a 4-byte instruction group. The instruction op-code loading the literal may reside in ANY of the other three bytes in the instruction group.

EXAMPLE

BYTE 1	BYTE 2	BYTE 3
LOAD-SHORT-LITERAL	QQQQQQQQ	QQQQQQQQ
BYTE 4		
00001111		

In this example QQQQQQQQ indicates any other 8-bit instruction. When Byte **1** is executed, binary 00001111 (HEX 0f) from Byte **4** will be pushed (right justified and padded by leading zeros) onto the Parameter Stack. Then the instructions in Byte **2** and Byte **3** will execute. The microprocessor instruction decoder knows not to execute Byte **4**. It is possible to push three identical 8-bit values as follows:

BYTE 1	BYTE 2
LOAD-SHORT-LITERAL	LOAD-SHORT-LITERAL
BYTE 3	BYTE 4
LOAD-SHORT-LITERAL	00001111

SHORT-LITERAL-INSTRUCTION

LOAD-SHORT-LITERAL—Push the 8-bit value found in Byte **4** of the current 4-byte instruction group onto the Parameter Stack

LOGIC INSTRUCTIONS

Logical and math operations use the stack for the source of one or two operands and as the destination for results. The stack organization is a particularly convenient arrangement for evaluating expressions. TOP indicates the top value on the Parameter Stack **74**. NEXT indicates the next to top value on the Parameter Stack **74**.

AND—Pop TOP and NEXT from the Parameter Stack, perform the logical AND operation on these two operands, and push the result onto the Parameter Stack.

OR—Pop TOP and NEXT from the Parameter Stack, perform the logical OR operation on these two operands and push the result onto the Parameter Stack

XOR—Pop TOP and NEXT from the Parameter Stack, perform the logical exclusive OR on these two operands, and push the result onto the Parameter Stack

BIT-CLEAR—Pop TOP and NEXT from the Parameter Stack, toggle all bits in NEXT, perform the logical AND operation on TOP, and push the result onto the Parameter Stack (Another way of understanding this instruction is thinking of it as clearing all bits in TOP that are set in NEXT)

MATH INSTRUCTIONS

Math instruction pop the TOP item and NEXT to top item of the Parameter Stack **74** to use as the operands. The results are pushed back on the Parameter Stack. The CARRY flag is used to latch the "33rd bit" of the ALU result

ADD—Pop the TOP item and NEXT to top item from the Parameter Stack, add the values together and push the result back on the Parameter Stack. The CARRY flag may be changed

ADD-WITH-CARRY—Pop the TOP item and the NEXT to top item from the Parameter Stack, add the values together. If the CARRY flag is "1" increment the result. Push the ultimate result back on the Parameter Stack. The CARRY flag may be changed

ADD-X—Pop the TOP item from the Parameter Stack and read the third item from the top of the Parameter Stack. Add the values together and push the result back on the Parameter Stack. The CARRY flag may be changed

SUB—Pop the TOP item and NEXT to top item from the Parameter Stack, Subtract NEXT from TOP and push the result back on the Parameter Stack. The CARRY flag may be changed

SUB-WITH-CARRY—Pop the TOP item and NEXT to top item from the Parameter Stack. Subtract NEXT from TOP. If the CARRY flag is "1" increment the result. Push the ultimate result back on the Parameter Stack. The CARRY flag may be changed.

SUB-X—

SIGNED-MULT-STEP—

UNSIGNED-MULT-STEP—

SIGNED-FAST-MULT—

FAST-MULT-STEP—

UNSIGNED-DIV-STEP—

GENERATE-POLYNOMIAL —

ROUND—

COMPARE—Pop the TOP item and NEXT to top item from the Parameter Stack. Subtract NEXT from TOP. If the result has the most significant bit equal to "0" (the result is positive), push the result onto the Parameter Stack. If the result has the most significant bit equal to "1" (the result is negative), push the old value of TOP onto the Parameter Stack. The CARRY flag may be affected

SHIFT/ROTATE

SHIFT-LEFT—Shift the TOP Parameter Stack item left one bit. The CARRY flag is shifted into the least significant bit of TOP

SHIFT-RIGHT—Shift the TOP Parameter Stack item right one bit. The least significant bit of TOP is shifted into the CARRY flag. Zero is shifted into the most significant bit of TOP

DOUBLE-SHIFT-LEFT—Treating the TOP item of the Parameter Stack as the most significant word of a 64-bit number and the NEXT stack item as the least significant word, shift the combined 64-bit entity left one bit. The CARRY flag is shifted into the least significant bit of NEXT

DOUBLE-SHIFT-RIGHT—Treating the TOP item of the Parameter Stack as the most significant word of a 64-bit number and the NEXT stack item as the least significant word, shift the combined 64-bit entity right one bit. The least significant bit of NEXT is shifted into the CARRY flag. Zero is shifted into the most significant bit of TOP

OTHER INSTRUCTIONS

FLUSH-STACK—Empty all on-chip Parameter Stack locations into off-chip RAM. (This instruction is useful for multitasking applications). This instruction accesses a counter which holds the depth of the on-chip stack and can require from none to 16 external memory cycles

FLUSH-RSTACK—Empty all on-chip Return Stack locations into off-chip RAM. (This instruction is useful for multitasking applications). This instruction accesses a counter which holds the depth of the on-chip Return Stack and can require from none to 16 external memory cycles.

It should further be apparent to those skilled in the art that various changes in form and details of the invention as

shown and described may be made. It is intended that such changes be included within the spirit and scope of the claims appended hereto.

What is claimed is:

1. A microprocessor integrated circuit comprising:

a program-controlled processing unit operative in accordance with a sequence of program instructions;

a memory coupled to said processing unit and capable of storing information provided by said processing unit;

a plurality of column latches coupled to the processing unit and the memory, wherein, during a read operation, a row of bits are read from the memory and stored in the column latch; and

a variable speed system clock having an output coupled to said processing unit;

said processing unit, said variable speed system clock, said plurality of column latches, and said memory fabricated on a single substrate, said memory using a greater area of said single substrate than said processing unit, said memory further using a majority of a total area of said single substrate.

2. The microprocessor integrated circuit of claim 1 wherein said memory is dynamic random-access memory.

3. The microprocessor integrated circuit of claim 1 wherein said memory is static random-access memory.

4. A microprocessor integrated circuit comprising:

a processing unit disposed upon an integrated circuit substrate, said processing unit operating in accordance with a predefined sequence of program instructions;

a memory coupled to said processing unit and capable of storing information provided by said processing unit, said memory occupying a larger area of said integrated circuit substrate than said processing unit said memory further occupying a majority of a total area of said single substrate; and

a ring oscillator having a variable output frequency, wherein the ring oscillator provides a system clock to the processing unit, the ring oscillator disposed on said integrated circuit substrate.

5. The microprocessor integrated circuit of claim 4 wherein said memory is dynamic random-access memory.

6. The microprocessor integrated circuit of claim 4 wherein said memory is static random-access memory.

7. The microprocessor integrated circuit of claim 4 wherein said memory is capable of supporting read and write operations.

8. A microprocessor integrated circuit comprising:

a processing unit having one or more interface ports for interprocessor communication, said processing unit being disposed on a single substrate;

a memory disposed upon said substrate and coupled to said processing unit, said memory occupying a greater area of said substrate than said processing unit, said memory further comprising a majority of a total area of said substrate; and

a ring oscillator having a variable output frequency, wherein the ring oscillator provides a system clock to the processing unit, the ring oscillator disposed on said substrate.

9. The microprocessor integrated circuit of claim 8 wherein a first of said interface ports includes a column latch, said column latch facilitating serial communication through said first of said interface ports.

10. The microprocessor integrated circuit of claim 8 further including memory controller means coupled to said memory for performing direct memory access data transfer through said one or more interface ports.

11. A microprocessor computational system comprising:

a first processing unit disposed upon a first substrate;

a first memory disposed upon said first substrate and coupled to said first processing unit, said first memory occupying a greater area of said first substrate than said first processing unit, said memory further occupying a majority of a total area of said substrate;

a ring oscillator having a variable output frequency, wherein the ring oscillator provides a system clock to the processing unit, the ring oscillator disposed on said first substrate; and

a second processing unit coupled to said first processing unit and configured for interprocessor communication with said first processing unit.

12. The microprocessor computational system of claim 11 wherein said second processing unit and a second memory are disposed upon a second substrate, said second memory occupying a greater area of said second substrate than said second processing unit, said second memory further occupying a majority of a total area of said substrate.

13. The multiprocessor computational system of claim 11 wherein said first processing unit includes an interface port for establishing said interprocessor communication between an internal register of said first processing unit and second processing unit.

* * * * *

The JS 44 civil cover sheet and the information contained herein neither replace nor supplement the filing and service of pleadings or other papers as required by law, except as provided by local rules of court This form, approved by the Judicial Conference of the United States in September 1974 is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet (SEE INSTRUCTIONS ON THE REVERSE OF THE FORM)

DF

I. (a) PLAINTIFFS

Technology Properties Limited and Patriot Scientific Corporation

(b) County of Residence of First Listed Plaintiff _____
(EXCEPT IN U S PLAINTIFF CASES)

(C) Attorney's (Firm Name, Address and Telephone Number)

S Calvin Capshaw
Capshaw DeRieux, LLP
1127 Judson Road, Suite 220
P O Box 3999 (75606-3999)
Longview, TX 75601
(903) 236-9800

DEFENDANTS
HTC Corporation and HTC America, Inc.

County of Residence of First Listed Defendant _____
(IN U.S. PLAINTIFF CASES ONLY)
NOTE IN LAND CONDEMNATION CASES USE THE LOCATION OF THE LAND INVOLVED

Attorneys (If Known)

2-08 CV-172

II. BASIS OF JURISDICTION (PLACE AN 'X" IN ONE BOX ONLY)

- ☐ 1 U S. Government Plaintiff
- ☒ 3 Federal Question (U S Government Not a Party)
- ☐ 2 U S. Government Defendant
- ☐4 Diversity (Indicate Citizenship of Parties in Item III)

III. CITIZENSHIP OF PRINCIPAL PARTIES (Place an "X" in One Box for Plaintiff and One Box for Defendant)
(For Diversity Cases Only)

	PTF	DEF		PTF	DEF
Citizen of This State	☐ 1	☐ 1	Incorporated or Principal Place of Business in This State	☐4	☐ 4
Citizen of Another State	☐2	☐ 2	Incorporated and Principal Place of Business in Another State	☐5	☐ 5
Citizen or Subject of a Foreign Nation	☐ 3	☐ 3	Foreign Country	☐ 6	☐ 6

IV. NATURE OF SUIT (Place an 'X' in One Box Only)

CONTRACT
- ☐ 110 Insurance
- ☐ 120 Marine
- ☐ 130 Miller Act
- ☐ 140 Negotiable Instrument
- ☐ 150 Recovery of Overpayment & Enforcement of Judgment
- ☐ 151 Medicare Act
- ☐ 152 Recovery of Defaulted Student Loans (Excl Veterans)
- ☐ 153 Recovery of Overpayment of Veteran's Benefits
- ☐ 160 Stockholders' Suits
- ☐ 190 Other Contract
- ☐ 195 Contract Product Liability
- ☐ 196 Franchise

REAL PROPERTY
- ☐ 210 Land Condemnation
- ☐ 220 Foreclosure
- ☐ 230 Rent Lease & Ejectment
- ☐ 240 Torts to Land
- ☐ 245 Tort Product Liability
- ☐ 290 All Other Real Property

TORTS

PERSONAL INJURY
- ☐ 310 Airplane
- ☐ 315 Airplane Product Liability
- ☐ 320 Assault Libel & Slander
- ☐ 330 Federal Employers Liability
- ☐ 340 Marine
- ☐ 345 Marine Product Liability
- ☐ 350 Motor Vehicle
- ☐ 355 Motor Vehicle Product Liability
- ☐ 360 Other Personal Injury

PERSONAL INJURY
- ☐ 362 Personal Injury - Med. Malpractice
- ☐ 365 Personal Injury – Product Liability
- ☐ 368 Asbestos Personal Injury Product Liability

PERSONAL PROPERTY
- ☐ 370 Other Fraud
- ☐ 371 Truth in Lending
- ☐ 380 Other Personal Property Damage
- ☐ 385 Property Damage Product Liability

CIVIL RIGHTS
- ☐ 441 Voting
- ☐ 442 Employment
- ☐ 443 Housing Accommodations
- ☐ 444 Welfare
- ☐ 440 Other Civil Rights

PRISONER PETITIONS
- ☐ 510 Motions to Vacate Sentence

HABEAS CORPUS:
- ☐ 530 General
- ☐ 535 Death Penalty
- ☐ 540 Mandamus & Other
- ☐ 550 Civil Rights
- ☐ 555 Prison Condition

FORFEITURE/PENALTY
- ☐ 610 Agriculture
- ☐ 620 Other Food & Drug
- ☐ 625 Drug Related Seizure of Property 21 USC 881
- ☐ 630 Liquor Laws
- ☐ 640 R R & Truck
- ☐ 650 Airline Regs
- ☐ 660 Occupational Safety/Health
- ☐ 690 Other

LABOR
- ☐ 710 Fair Labor Standards Act
- ☐ 720 Labor/Mgmt Relations
- ☐ 730 Labor/Mgmt. Reporting & Disclosure Act
- ☐ 740 Railway Labor Act
- ☐ 790 Other Labor Litigation
- ☐ 791 Empl. Ret. Inc Security Act

BANKRUPTCY
- ☐ 422 Appeal 28 USC 158
- ☐ 423 Withdrawal 28 USC 157

PROPERTY RIGHTS
- ☐ 820 Copyrights
- x 830 Patent
- ☐ 840 Trademark

SOCIAL SECURITY
- ☐ 861 HIA (1 395ff)
- ☐ 862 Black Lung (923)
- ☐ 863 DIWC/DIWW (405(g))
- ☐ 864 SSID Title XVI
- ☐ 865 RSI (405(g))

FEDERAL TAX SUITS
- ☐ 870 Taxes (U.S. Plaintiff or Defendant
- ☐ 871 IRS – Third Party 26 USC 7609

OTHER STATUTES
- ☐ 400 State Reapportionment
- ☐ 410 Antitrust
- ☐ 430 Banks and Banking
- ☐ 450 Commerce
- ☐ 460 Deportation
- ☐ 470 Racketeer Influenced and Corrupt Organizations
- ☐ 480 Consumer Credit
- ☐ 490 Cable/Sat TV
- ☐ 810 Selective Service
- ☐ 850 Securities/Commodities/ Exchange
- ☐ 875 Customer Challenge 12 USC 3410
- ☐ 890 Other Statutory Actions
- ☐ 891 Agricultural Acts
- ☐ 892 Economic Stabilization Act
- ☐ 893 Environmental Matters
- ☐ 894 Energy Allocation Act
- ☐ 895 Freedom of Information Act
- ☐ 900 Appeal of Fee Determination Under Equal Access to Justice
- ☐ 950 Constitutionality of State State Statutes

V. ORIGIN (Place an "X" in One Box Only)

☒1 Original Proceeding	☐ 2 Removed from State Court	☐ 3 Remanded from Appellate Court	☐ 4 Reinstated or Reopened	☐ 5 Transferred from another district (specify)	☐ 6 Multidistrict Litigation	☐ 7 Appeal to District Judge from Magistrate Judgment

VI. CAUSE OF ACTION (Cite the U.S Civil Statute Under which you are filing (Do not cite jurisdictional statutes unless diversity): **35 U.S.C. § 271**
(Brief description of cause:

VII. REQUESTED IN COMPLAINT: ☐ CHECK IF THIS IS A CLASS ACTION UNDER FR C P 23 DEMAND $ CHECK YES only if demanded in complaint: JURY DEMAND: ☒ YES ☐ NO

VII. RELATED CASE(S) (See Instructions): **IF ANY** See the attached page

DATE April 25, 2008

SIGNATURE OF ATTORNEY OF RECORD

FOR OFFICE USE ONLY

RECEIPT # _____ AMOUNT _____ APPLYING IFP _____ JUDGE _____ MAG. JUDGE _____

Related Cases

1. Technology Properties Limited, Inc , et al v Fujitsu Limited, et al; In the United States District Court for the Eastern District of Texas, Marshall Division; Cause No 2:05-cv-00494 (TJW)

2. Technology Properties Limited, Inc , et al v. HTC Corporation, et al; In the United States District Court for the Eastern District of Texas, Marshall Division - Cause No 2:08-cv-_____ - No cause number yet – filed simultaneously on April 25, 2008

3. Technology Properties Limited, Inc , et al v HTC Corporation, et al; In the United States District Court for the Eastern District of Texas, Marshall Division - Cause No 2:08-cv-_____ - No cause number yet – filed simultaneously on April 25, 2008.

4. Technology Properties Limited, Inc , et al v. ASUSTek Computer, Inc ; In the United States District Court for the Eastern District of Texas, Marshall Division - Cause No 2:08-cv-_____ - No cause number yet – filed simultaneously on April 25, 2008.

5. Technology Properties Limited, Inc , et al v. ASUSTek Computer, Inc ; In the United States District Court for the Eastern District of Texas, Marshall Division - Cause No. 2:08-cv-_____ - No cause number yet – filed simultaneously on April 25, 2008.

6. Technology Properties Limited, Inc , et al v Acer, Inc , et al; In the United States District Court for the Eastern District of Texas, Marshall Division; Cause No 2:08-cv-_____ - No cause number yet – filed simultaneously on April 25, 2008.

7. Technology Properties Limited, Inc , et al v. Acer, Inc , et al; In the United States District Court for the Eastern District of Texas, Marshall Division; Cause No 2:08-cv-_____ - No cause number yet – filed simultaneously on April 25, 2008

EXHIBIT F

IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

(1) TECHNOLOGY PROPERTIES LIMITED and (2) PATRIOT SCIENTIFIC CORPORATION,

Plaintiffs,

vs.

(1) HTC CORPORATION
(2) HTC AMERICA, INC. and
(3) ASUSTeK COMPUTER, INC.,

Defendants.

CASE NO. 2-08CV-174 TJW

Jury Trial Demanded

COMPLAINT FOR PATENT INFRINGEMENT AND DEMAND FOR JURY TRIAL

Plaintiffs, Technology Properties Limited ("TPL") and Patriot Scientific Corporation ("Patriot"), (collectively "Plaintiffs"), allege the following in support of their Complaint for Patent Infringement and Demand for Jury Trial ("Complaint") against Defendants, HTC Corporation ("HTC"), HTC America, Inc. ("HTC America"), and ASUSTeK Computer, Inc ("ASUSTeK")

PARTIES

1. Plaintiff, Technology Properties Limited ("TPL") is a corporation duly organized and existing under the laws of the State of California and maintains its principal place of business in Cupertino, California

2. Plaintiff, Patriot Scientific Corporation ("Patriot") is a corporation duly organized and existing under the laws of the State of Delaware and maintains its principal place

of business in Carlsbad, California.

3. Upon information and belief, Defendant HTC Corporation is a Taiwan corporation with its principal place of business in Taoyuan, Taiwan, R.O.C.

4. Upon information and belief, Defendant HTC America, Inc. is a Texas corporation with its principal place of business in Bellevue, Washington.

5. Upon information and belief, Defendant ASUSTeK Computer, Inc. is a Taiwan corporation with its principal place of business in Taipei, Taiwan, R.O.C.

JURISDICTION

6. This Court has subject matter jurisdiction over this action pursuant to 28 U.S.C. §§ 1331, 1338(a) because this action arises under the patent laws of the United States, including 35 U.S.C. §§ 101, *et seq.* and 271, *et seq.* This Court has personal jurisdiction over Defendants because they each infringe Plaintiffs' patent by offering on their websites infringing products to their users and/or customers who reside in, or may be found in, the Eastern District of Texas. Further, each Defendant has actually transacted business with users of their websites in the Eastern District of Texas.

VENUE

7. Venue is proper in this judicial district under 28 U.S.C. §§ 1391(b) and 1400(b) because Defendants reside in this district, have each committed acts of infringement in this district and, through their websites, have a regular and established place of business in this district.

GENERAL ALLEGATIONS

8. On July 21, 1998, United States Patent No. 5,784,584 ('584 Patent") entitled "High Performance Microprocessor Using Instructions That Operate Within Instruction

Groups" was duly and legally issued. All rights and interest in the '584 Patent are co-owned by TPL and Patriot. TPL has the sole and exclusive right and obligation to license and enforce the '584 Patent. A true and correct copy of the '584 Patent is attached hereto as Exhibit A.

COUNT 1

(Patent Infringement Against HTC Corporation)

9. Paragraphs 1-8 of the Complaint set forth above are incorporated herein by reference.

10. Upon information and belief Defendant HTC has infringed and continues to infringe under 35 U.S.C. § 271 United States Patent No. 5,784,584.

11. HTC's acts of infringement have caused damage to Plaintiffs. Under 35 U.S.C. § 284, Plaintiffs are entitled to recover from HTC the damages sustained by Plaintiffs as a result of its infringement of the '584 Patent. HTC's infringement of Plaintiffs' exclusive rights under the '584 Patent will continue to damage Plaintiffs' business, causing irreparable harm, for which there is no adequate remedy at law, unless enjoined by this Court under 35 U.S.C. § 283.

12. Plaintiffs allege, on information and belief, that HTC's acts of infringement were willful and deliberate.

COUNT 2

(Patent Infringement Against HTC America, Inc.)

13. Paragraphs 1-8 of the Complaint set forth above are incorporated herein by reference.

14. Upon information and belief Defendant HTC America has infringed and continues to infringe under 35 U.S.C. § 271 United States Patent No. 5,784,584.

15. HTC America's acts of infringement have caused damage to Plaintiffs. Under 35

U.S.C. § 284, Plaintiffs are entitled to recover from HTC America the damages sustained by Plaintiffs as a result of its infringement of the '584 Patent. HTC America's infringement of Plaintiffs' exclusive rights under the '584 Patent will continue to damage Plaintiffs' business, causing irreparable harm, for which there is no adequate remedy at law, unless enjoined by this Court under 35 U.S.C. § 283.

16. Plaintiffs allege, on information and belief, that HTC America's acts of infringement were willful and deliberate.

COUNT 3

(Patent Infringement Against ASUSTeK Computer, Inc.)

17. Paragraphs 1-8 of the Complaint set forth above are incorporated herein by reference.

18. Upon information and belief Defendant ASUSTeK has infringed and continues to infringe under 35 U.S.C. § 271 United States Patent No. 5,784,584.

19. ASUSTeK's acts of infringement have caused damage to Plaintiffs. Under 35 U.S.C. § 284, Plaintiffs are entitled to recover from ASUSTeK the damages sustained by Plaintiffs as a result of its infringement of the '584 Patent. ASUSTeK's infringement of Plaintiffs' exclusive rights under the '584 Patent will continue to damage Plaintiffs' business, causing irreparable harm, for which there is no adequate remedy at law, unless enjoined by this Court under 35 U.S.C. § 283.

20. Plaintiffs allege, on information and belief, that ASUSTeK's acts of infringement were willful and deliberate.

PRAYER FOR RELIEF

WHEREFORE, Plaintiffs respectfully request that this Court enter judgment against

Defendants as follows:

A For judgment that Defendants HTC Corporation, HTC America, Inc. and ASUSTeK Computer, Inc have infringed and continue to infringe the '584 patent;

B. For permanent injunctions under 35 U.S.C. § 283 against Defendants and their directors, officers, employees, agents, subsidiaries, parents, attorneys, and all persons acting in concert, on behalf of, in joint venture, or in partnership with Defendants from further acts of infringement;

C For damages to be paid by Defendants adequate to compensate Plaintiffs for their infringement, including interests, costs and disbursements as the Court may deem appropriate under 35 U S C. § 284;

D. For judgment finding that Defendants infringement was willful and deliberate, entitling Plaintiffs to increased damages under 35 U S C § 284;

E. For judgment finding this to be an exceptional case against Defendants and awarding Plaintiffs attorney fees under 35 U.S.C § 285; and,

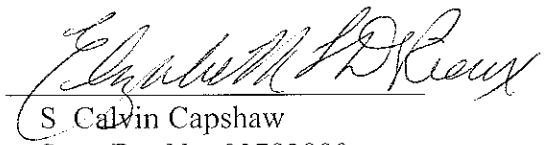
F For such other and further relief at law and in equity as the court may deem just and proper

DEMAND FOR JURY TRIAL

Pursuant to the Federal Rules of Civil Procedure Rule 38, Plaintiffs hereby demand a jury trial on all issues triable by jury

Dated: April 25, 2008

Respectfully submitted,

By: 

S. Calvin Capshaw
State Bar No. 03783900
Email: ccapshaw@capshawlaw.com
Elizabeth L. DeRieux
State Bar No. 05770585
Email: ederieux@capshawlaw.com
Capshaw DeRieux, LLP
1127 Judson Road, Suite 220
Longview, TX 75601
Telephone: (903) 236-9800
Facsimile: (903) 236-8787

Robert E. Krebs
California Bar No. 57526
Email: rkrebs@thelen.com
Christopher L. Ogden
California Bar No. 235517
Email: cogden@thelen.com
Thelen Reid Brown Raysman & Steiner, LLP
225 West Santa Clara Street, Suite 1200
San Jose, CA 95113-1723
Telephone: (408) 292-5800
Facsimile: (408) 287-8040

Ronald F. Lopez
California Bar No. 11756
Email: rflopez@thelen.com
Thelen Reid Brown Raysman & Steiner, LLP
101 Second Street, Suite 1800
San Francisco, CA 94105–3606
Telephone: (415) 371-1200
Facsimile: (415) 371-1211

ATTORNEYS FOR PLAINTIFF
TECHNOLOGY PROPERTIES LIMITED

By: Charles T. Hoge - by permission EFD

Robert M Parker
State Bar No 15498000
Email: rmparker@pbatyler.com
Robert Christopher Bunt
State Bar No. 00787165
Email: rcbunt@pbatyler.com
Parker, Bunt & Ainsworth, P.C.
100 East Ferguson, Ste. 1114
Tyler, TX 75702
Telephone: (903) 531-3535
Facsimile: (903) 533-9687

Charles T. Hoge
California Bar No. 110696
Email: choge@knlh.com
Kirby Noonan Lance & Hoge, LLP
350 Tenth Avenue, Suite 1300
San Diego, CA 92101
Telephone: (619) 231-8666
Facsimile: (619) 231-9593

ATTORNEYS FOR PLAINTIFF
PATRIOT SCIENTIFIC CORPORATION

US005784584A

United States Patent [19]

Moore et al.

[11] Patent Number: 5,784,584

[45] Date of Patent: Jul. 21, 1998

[54] **HIGH PERFORMANCE MICROPROCESSOR USING INSTRUCTIONS THAT OPERATE WITHIN INSTRUCTION GROUPS**

[75] Inventors: **Charles H. Moore**, Woodside; **Russell H. Fish, III** Mt. View, both of Calif

[73] Assignee: **Patriot Scientific Corporation**, San Diego, Calif

[21] Appl No: **484,935**

[22] Filed: **Jun. 7, 1995**

Related U.S. Application Data

[62] Division of Ser No 389,334, Aug 3, 1989 Pat No 5,440,749

[51] **Int. Cl.**[6] **G06F 9/30**

[52] **U.S. Cl.** **395/376**

[58] **Field of Search** 395/376, 382, 395/384, 588, 800.23

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,967,326 10/1990 May 395/800

5,127,091 6/1992 Bonfaral et al 395/375

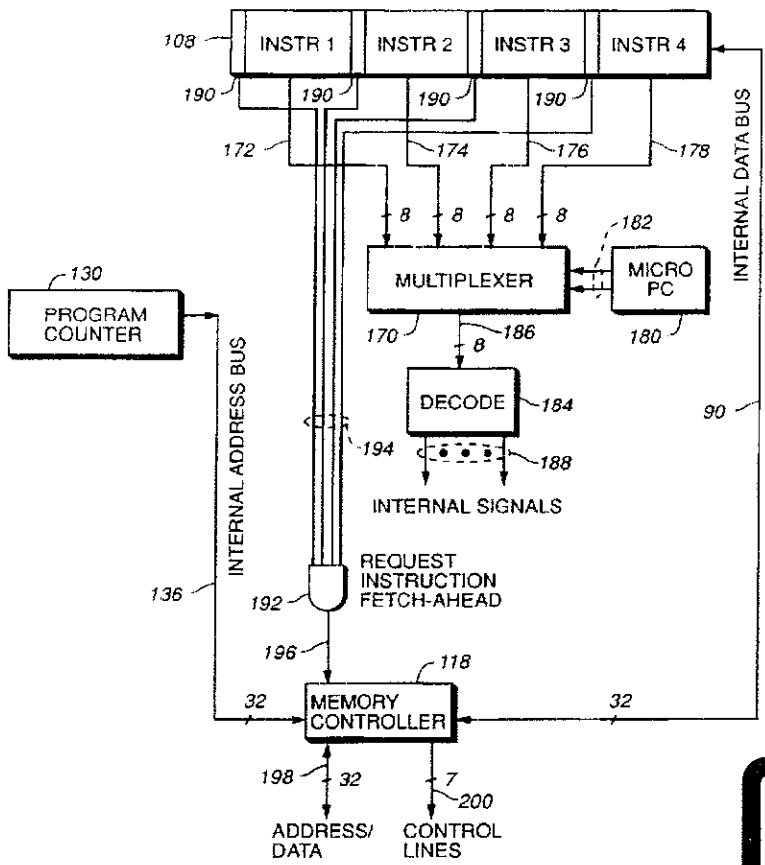
Primary Examiner—David Y Eng

Attorney, Agent, or Firm—Cooley Godward LLP

[57] **ABSTRACT**

A high-performance microprocessor system using instruction that access operands and instructions located relative to the current instruction group rather than located relative to the current instructions, as is the convention, is disclosed herein. The microprocessor system includes a central processing unit, memory, and a bus connecting the central processing unit and memory. An instruction fetching unit, connected to the bus, is provided for fetching instruction groups from the memory for use by the central processing unit and for storage within an instruction register. An instruction supplying unit operates to supply, in succession from the instruction register to the central processing unit, one or more instructions from each of the instruction groups. The system further includes an instruction decoder for configuring the instruction supplying unit to select, from the instruction register, operands associated with instructions from particular instruction groups.

29 Claims, 19 Drawing Sheets



EXHIBIT

tabbies

A

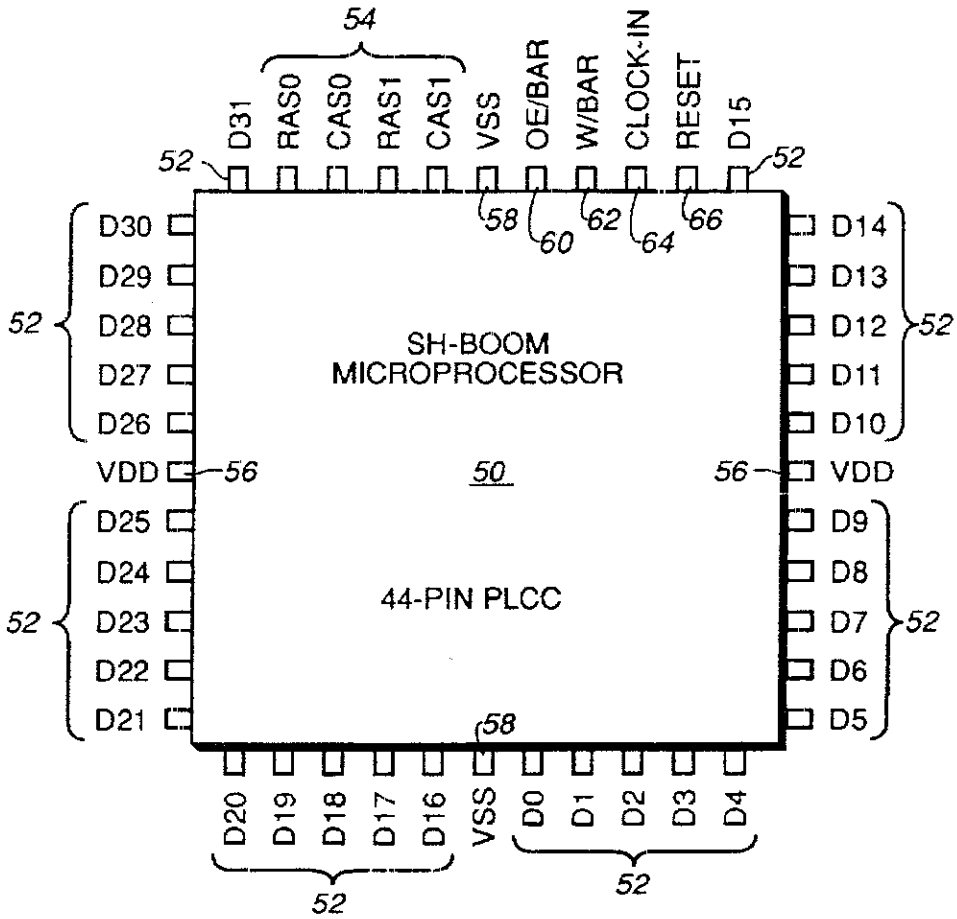

54
D31
RAS0
CAS0
RAS1
CAS1
VSS
OE/BAR
W/BAR
CLOCK-IN
RESET
D15
52
58
60
62
64
66
D30
D29
D28
D27
D26
VDD
D25
D24
D23
D22
D21
D14
D13
D12
D11
D10
VDD
D9
D8
D7
D6
D5
SH-BOOM
MICROPROCESSOR
50
44-PIN PLCC
56
D20
D19
D18
D17
D16
D0
D1
D2
D3
D4

FIG._1

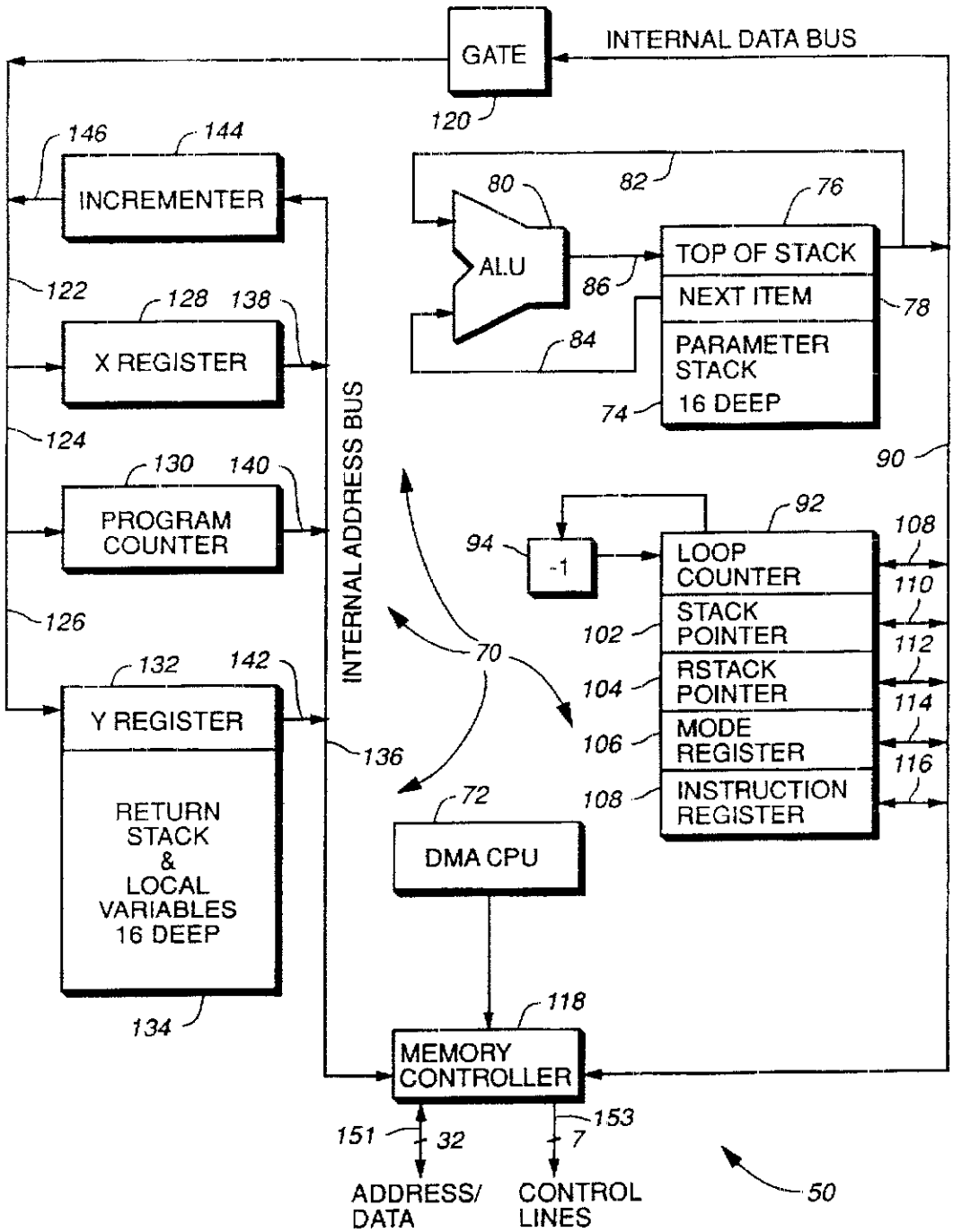

GATE
INTERNAL DATA BUS
120
146
144
INCREMENTER
80
82
76
ALU
TOP OF STACK
NEXT ITEM
PARAMETER STACK
16 DEEP
86
84
78
74
122
128
138
X REGISTER
124
130
140
PROGRAM COUNTER
126
132
142
Y REGISTER
RETURN STACK & LOCAL VARIABLES 16 DEEP
134
INTERNAL ADDRESS BUS
136
70
90
92
94
-1
LOOP COUNTER
STACK POINTER
RSTACK POINTER
MODE REGISTER
INSTRUCTION REGISTER
108
110
112
114
116
102
104
106
108
72
DMA CPU
118
MEMORY CONTROLLER
151
32
153
7
ADDRESS/ DATA
CONTROL LINES
50

FIG._2

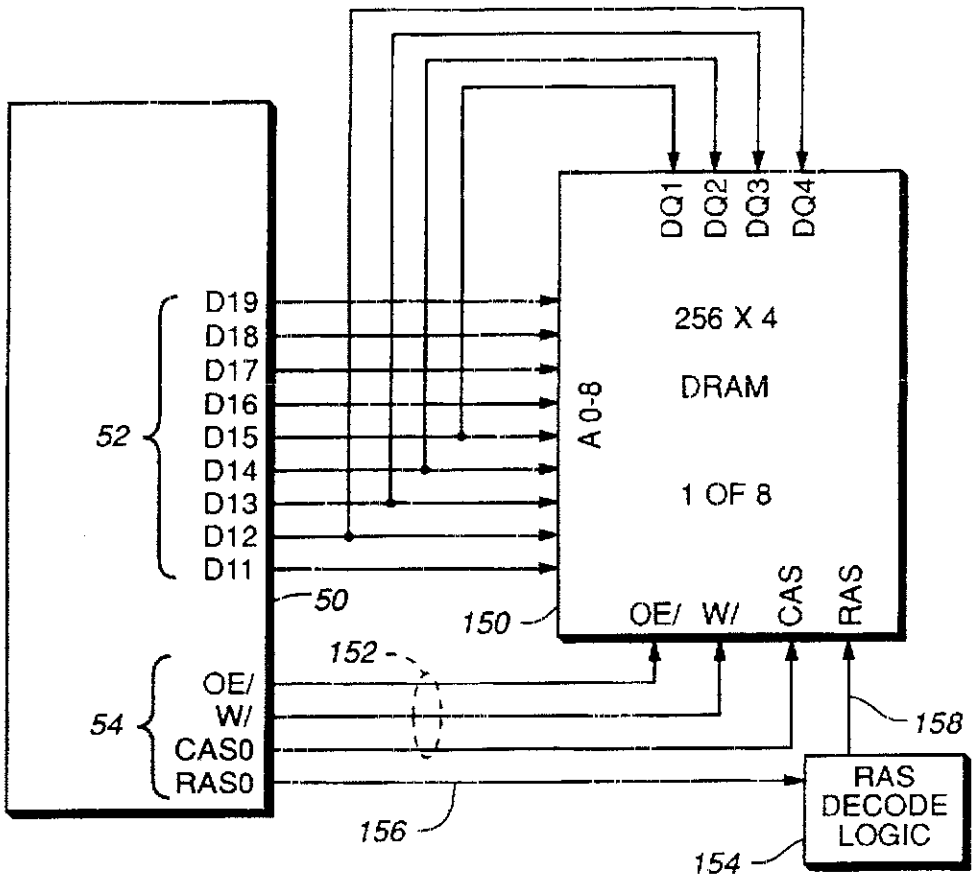

DQ1
DQ2
DQ3
DQ4
256 X 4
DRAM
1 OF 8
A 0-8
OE/
W/
CAS
RAS
D19
D18
D17
D16
D15
D14
D13
D12
D11
52
50
150
152
OE/
W/
CAS0
RAS0
54
156
158
RAS
DECODE
LOGIC
154

FIG._3

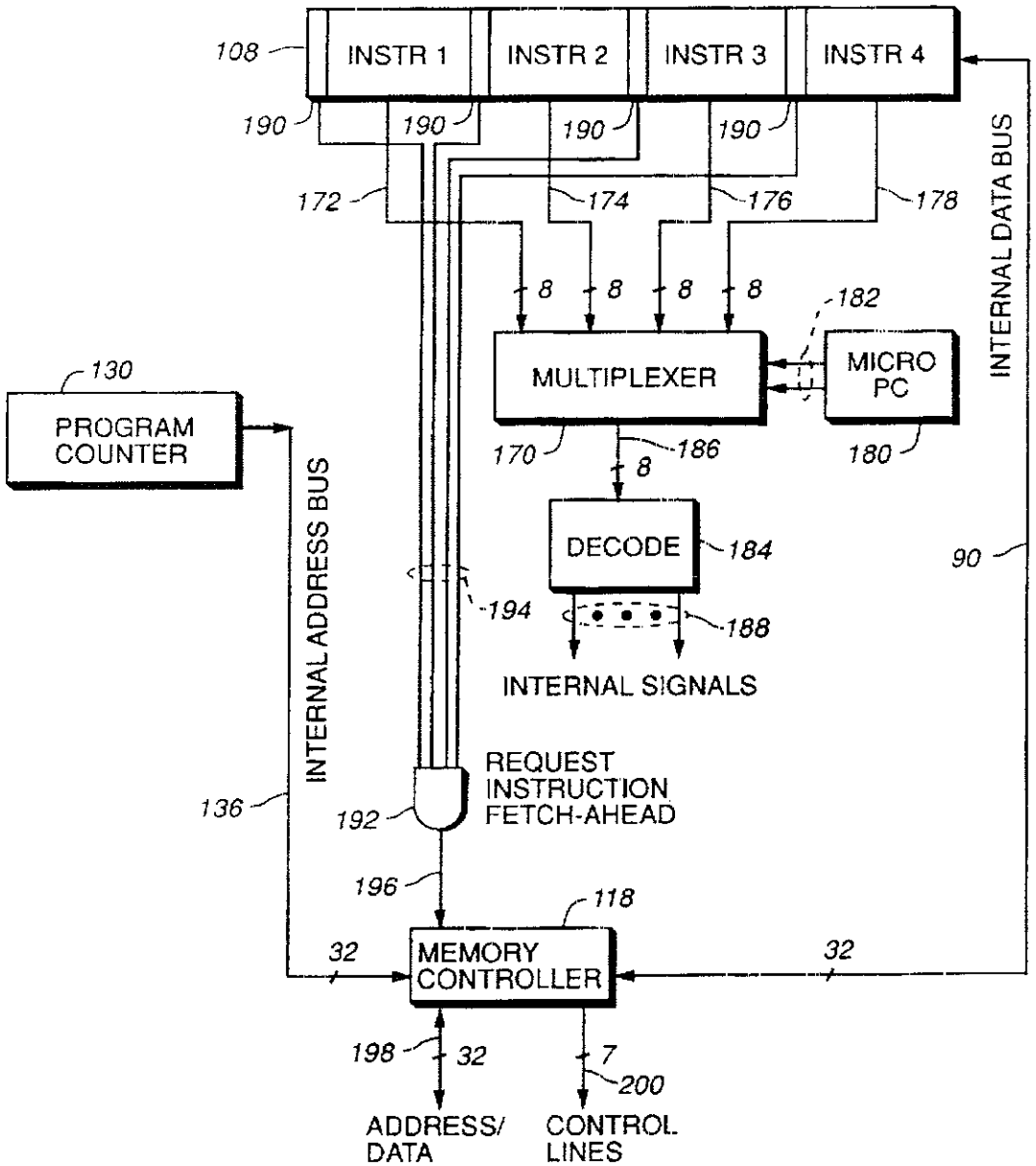

108
INSTR 1
INSTR 2
INSTR 3
INSTR 4
190
190
190
190
172
174
176
178
8
8
8
8
182
MULTIPLEXER
MICRO PC
INTERNAL DATA BUS
130
PROGRAM COUNTER
170
186
180
8
DECODE
184
90
194
188
INTERNAL SIGNALS
INTERNAL ADDRESS BUS
REQUEST INSTRUCTION FETCH-AHEAD
136
192
196
118
32
MEMORY CONTROLLER
32
198
32
7
200
ADDRESS/ DATA
CONTROL LINES

FIG._4

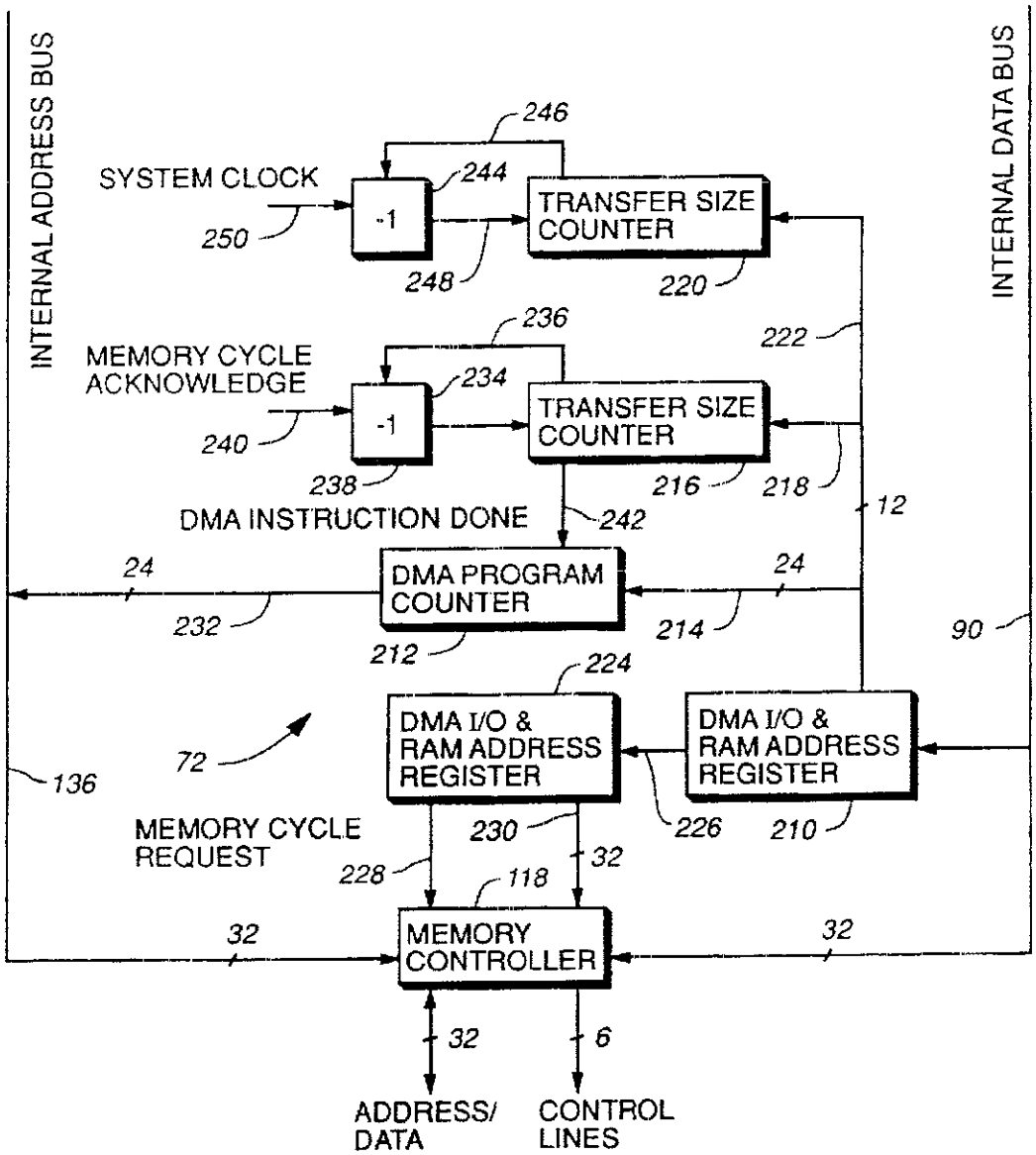

INTERNAL ADDRESS BUS
INTERNAL DATA BUS
SYSTEM CLOCK
246
244
250
-1
TRANSFER SIZE COUNTER
248
220
MEMORY CYCLE ACKNOWLEDGE
236
234
240
-1
TRANSFER SIZE COUNTER
238
216
218
222
12
DMA INSTRUCTION DONE
242
24
DMA PROGRAM COUNTER
24
232
214
212
90
224
DMA I/O & RAM ADDRESS REGISTER
DMA I/O & RAM ADDRESS REGISTER
72
136
MEMORY CYCLE REQUEST
230
226
210
228
32
118
32
MEMORY CONTROLLER
32
32
6
ADDRESS/ DATA
CONTROL LINES

FIG._5

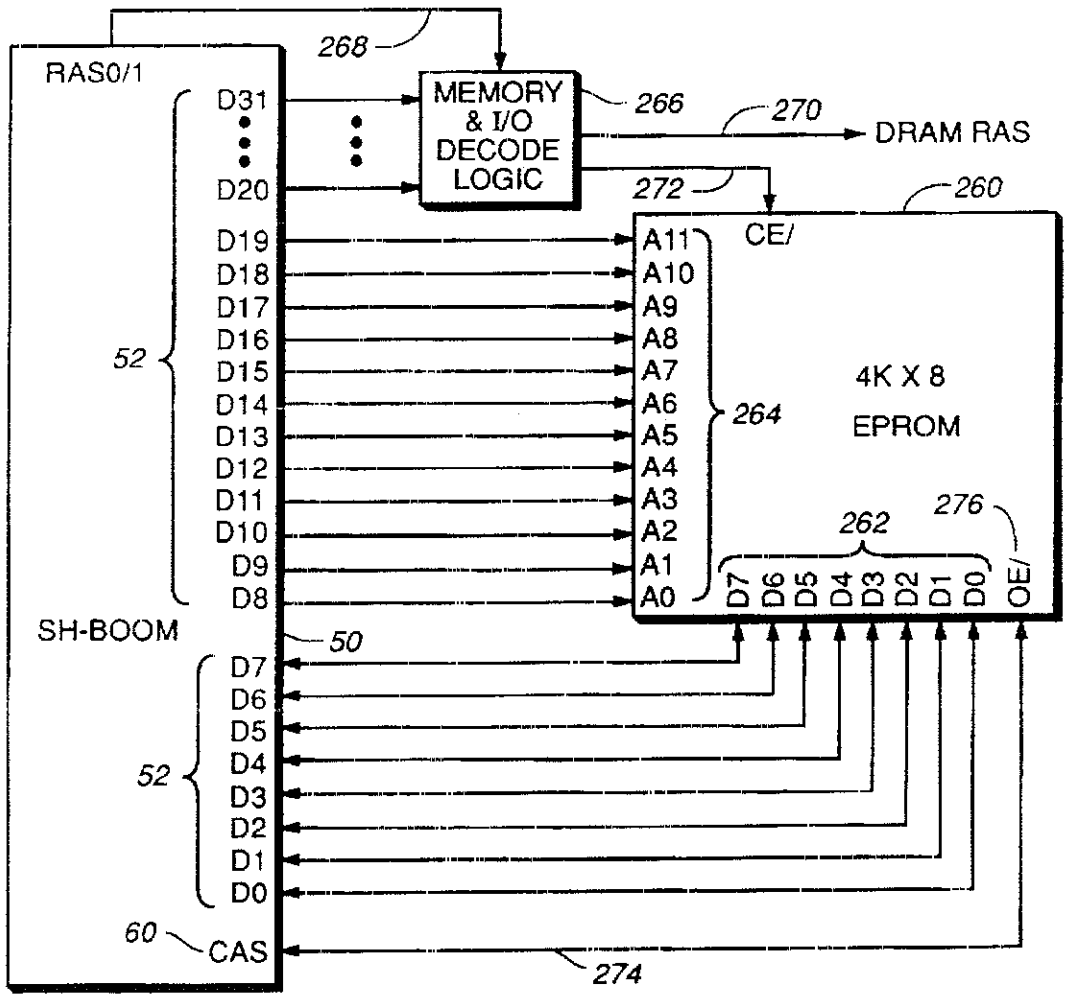

268
RAS0/1
D31
D20
MEMORY & I/O DECODE LOGIC
266
270
DRAM RAS
272
260
D19
D18
D17
D16
D15
D14
D13
D12
D11
D10
D9
D8
52
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
264
CE/
4K X 8
EPROM
262
276
D7
D6
D5
D4
D3
D2
D1
D0
OE/
SH-BOOM
50
D7
D6
D5
D4
D3
D2
D1
D0
52
60
CAS
274

FIG._6

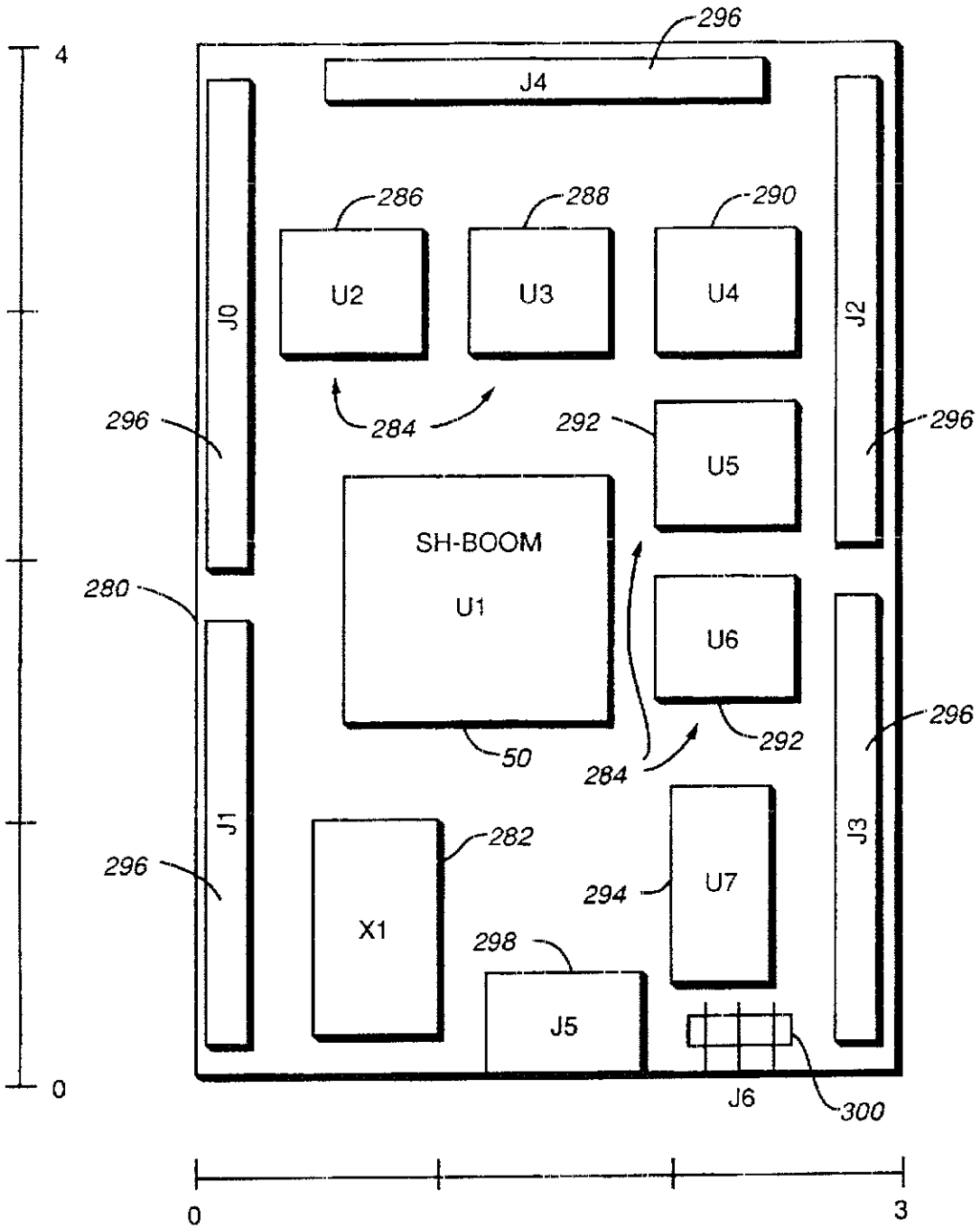

296
4
J4
J0
J2
286
288
290
U2
U3
U4
284
296
292
296
U5
SH-BOOM
U1
280
U6
296
50
284
292
J1
J3
296
282
294
U7
X1
298
J5
J6
300
0
0
3

FIG._7

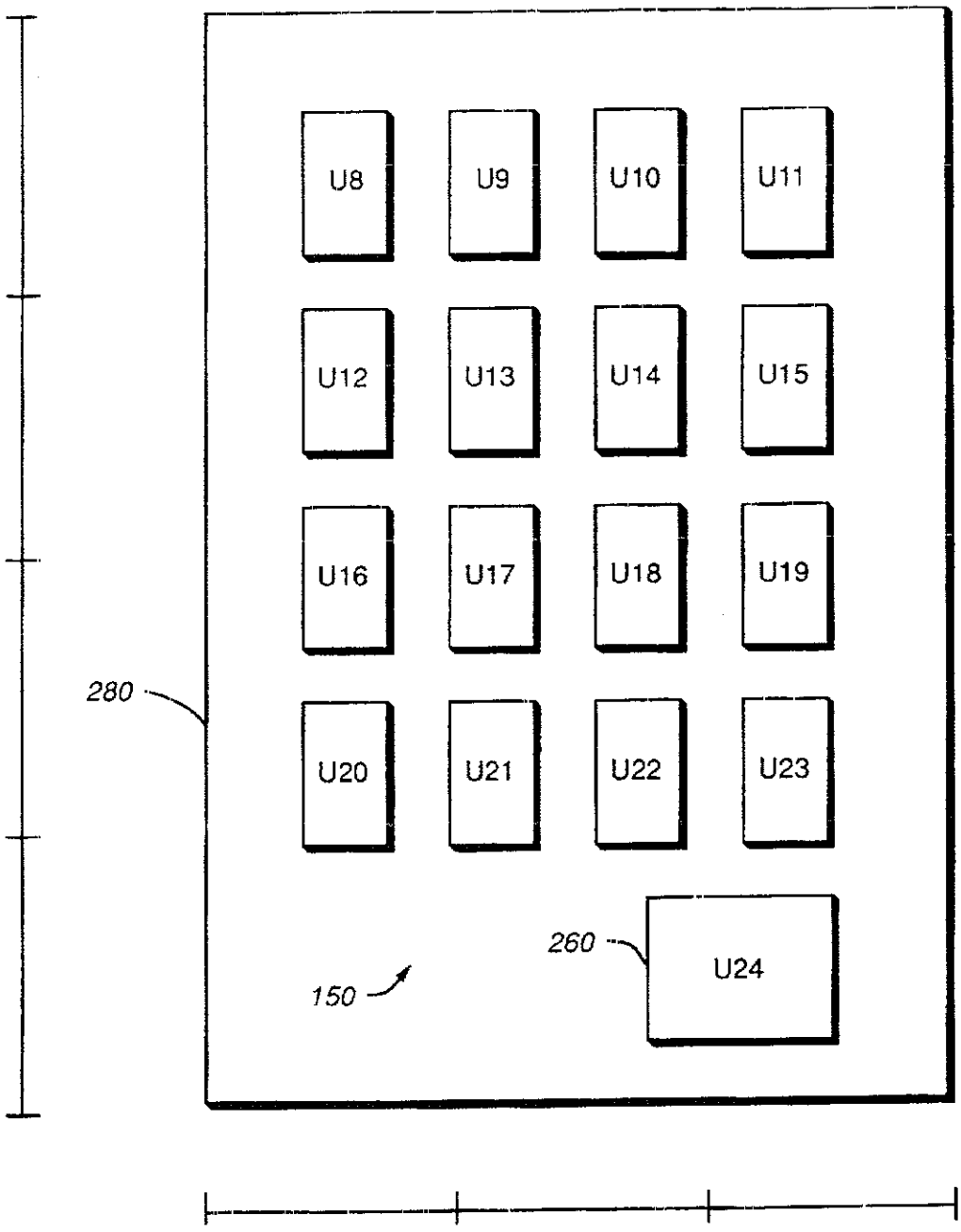

U8
U9
U10
U11
U12
U13
U14
U15
U16
U17
U18
U19
U20
U21
U22
U23
U24
280
260
150

FIG._8

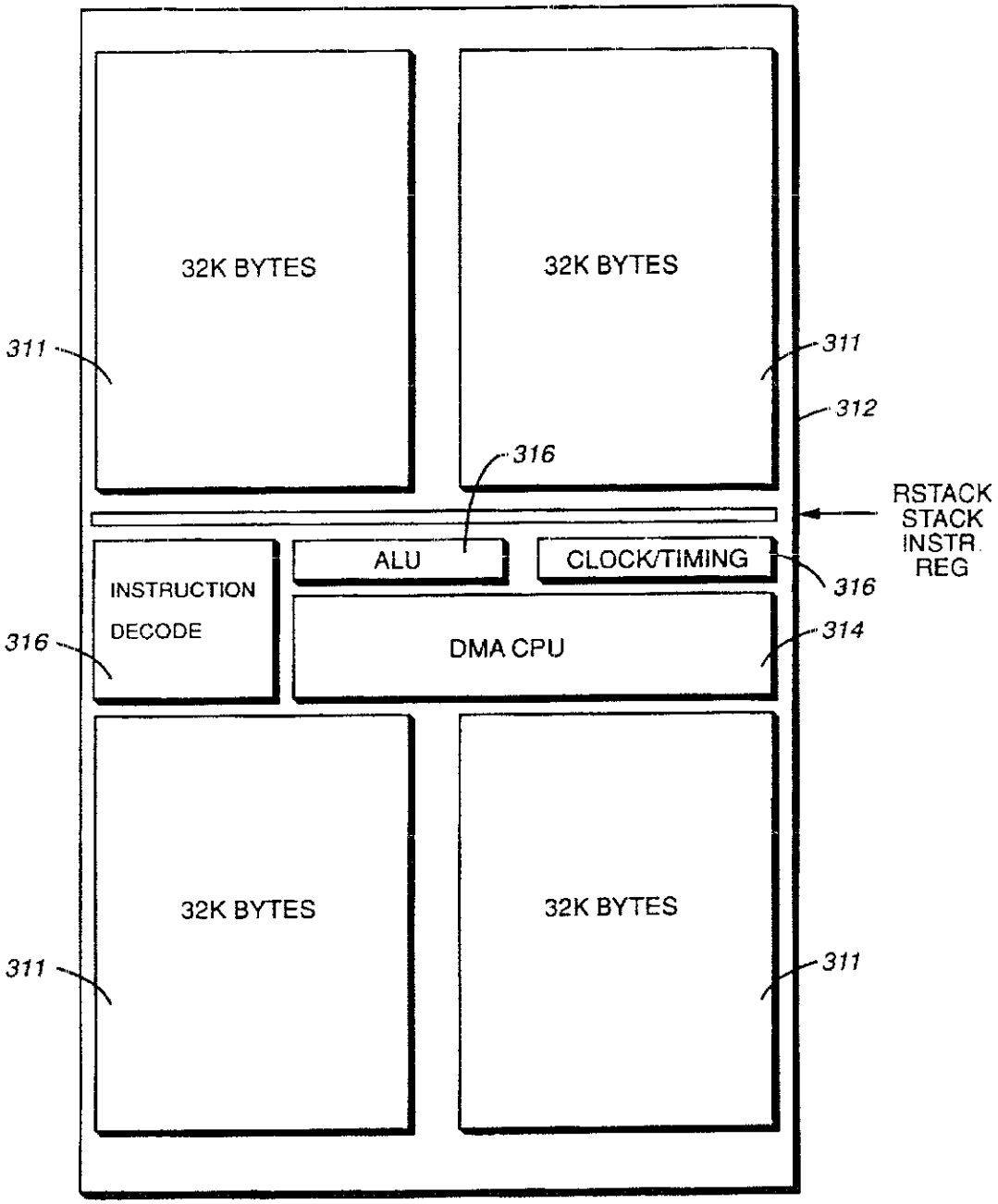

32K BYTES
32K BYTES
311
311
312
316
RSTACK
STACK
INSTR.
REG
ALU
CLOCK/TIMING
316
INSTRUCTION
DECODE
316
DMA CPU
314
32K BYTES
32K BYTES
311
311

FIG._9

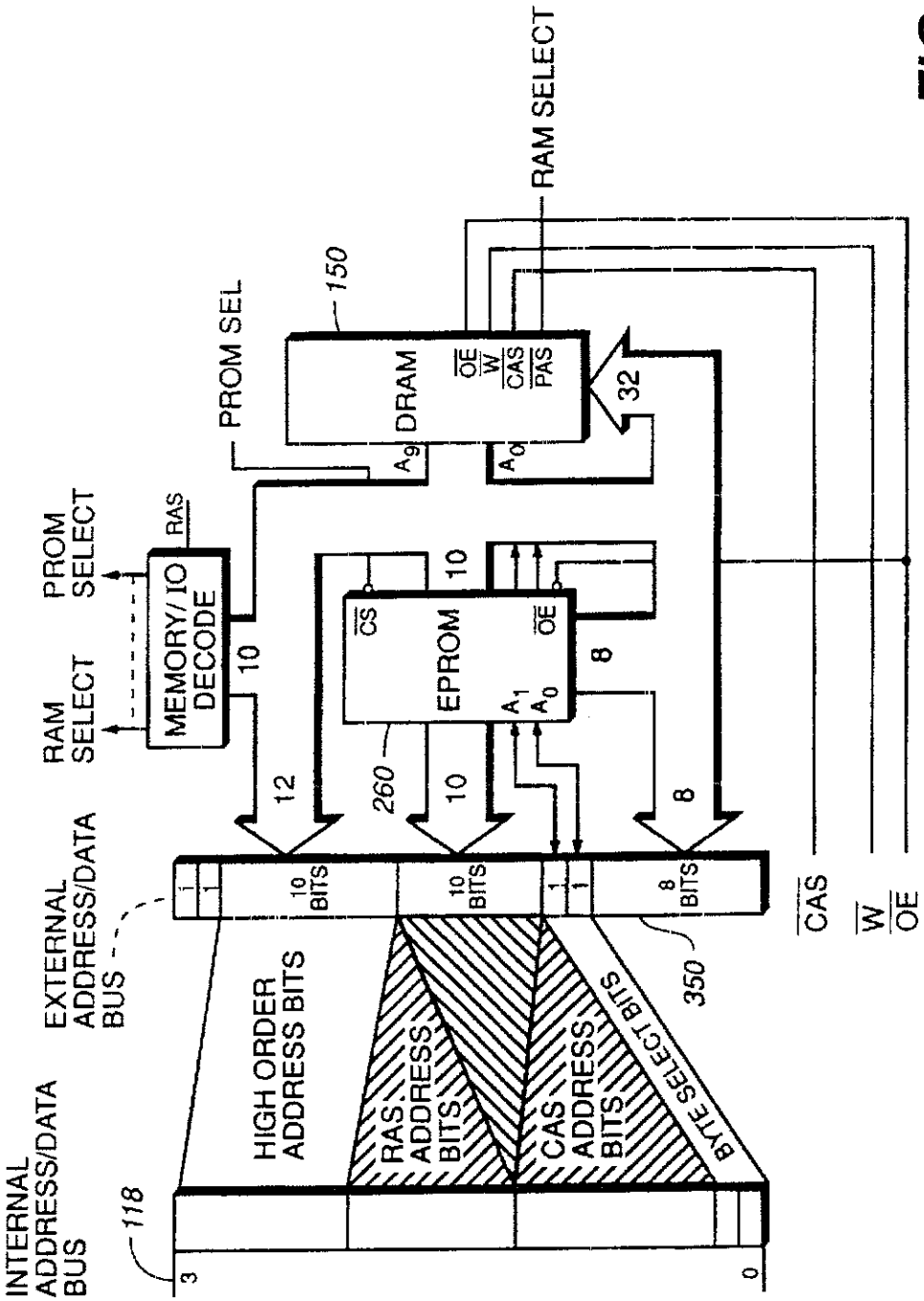
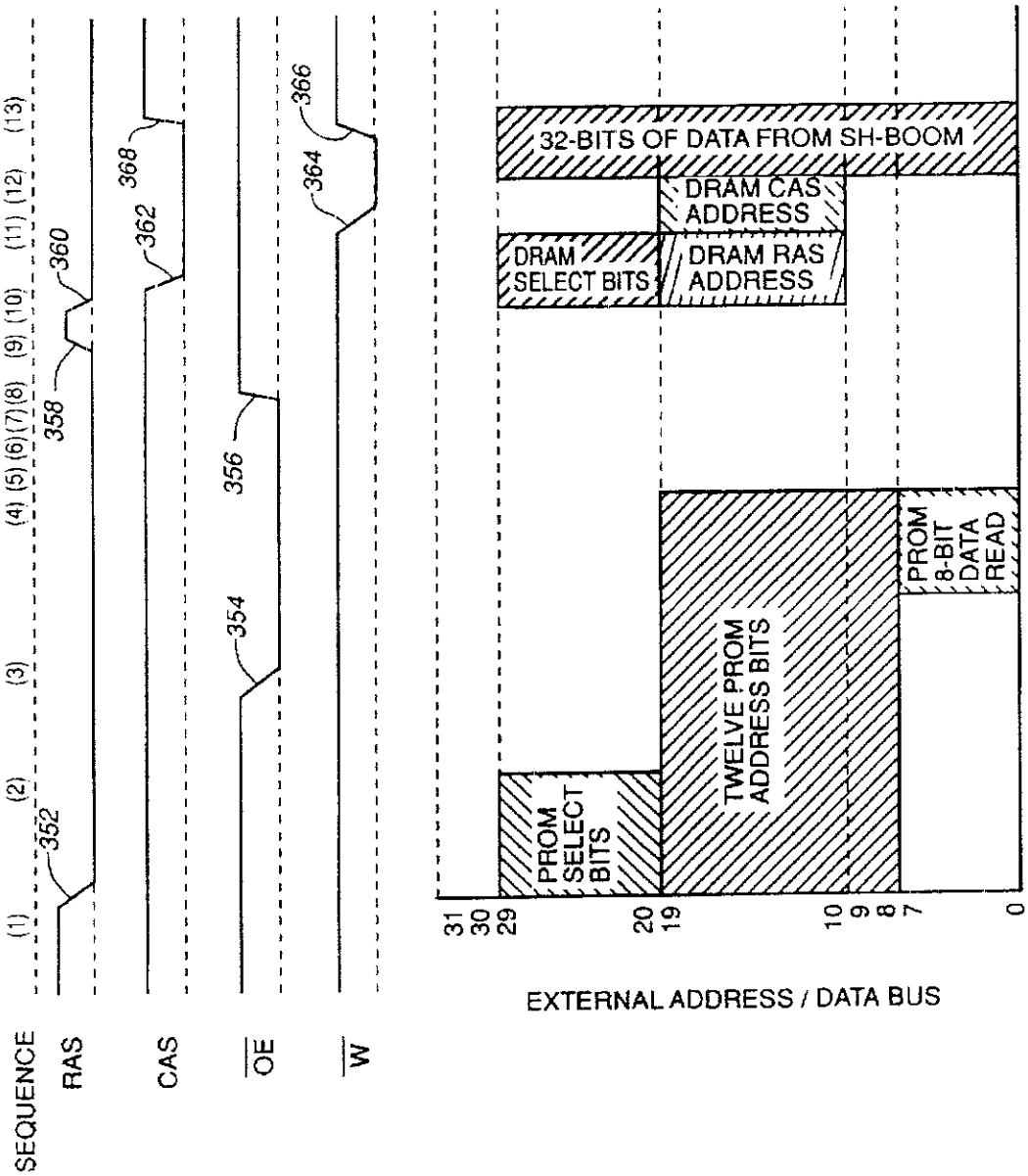

RAM SELECT
PROM SEL
150
DRAM
OE
W
CAS
RAS
32
A9
A0
PROM SELECT
RAS
MEMORY/IO DECODE
10
RAM SELECT
CS
EPROM
OE
A1
A0
10
8
260
12
10
8
EXTERNAL ADDRESS/DATA BUS
1
1
10 BITS
10 BITS
1
1
8 BITS
CAS
W
OE
350
HIGH ORDER ADDRESS BITS
RAS ADDRESS BITS
CAS ADDRESS BITS
BYTE SELECT BITS
INTERNAL ADDRESS/DATA BUS
118
3
0

FIG._10

FIG._11


SEQUENCE
RAS
CAS
OE
W
(1)
(2)
(3)
(4) (5) (6) (7) (8)
(9) (10)
(11) (12)
(13)
352
354
356
358
360
362
364
366
368
32-BITS OF DATA FROM SH-BOOM
DRAM CAS ADDRESS
DRAM SELECT BITS
DRAM RAS ADDRESS
PROM SELECT BITS
TWELVE PROM ADDRESS BITS
PROM 8-BIT DATA READ
31 30 29 20 19 10 9 8 7 0
EXTERNAL ADDRESS / DATA BUS

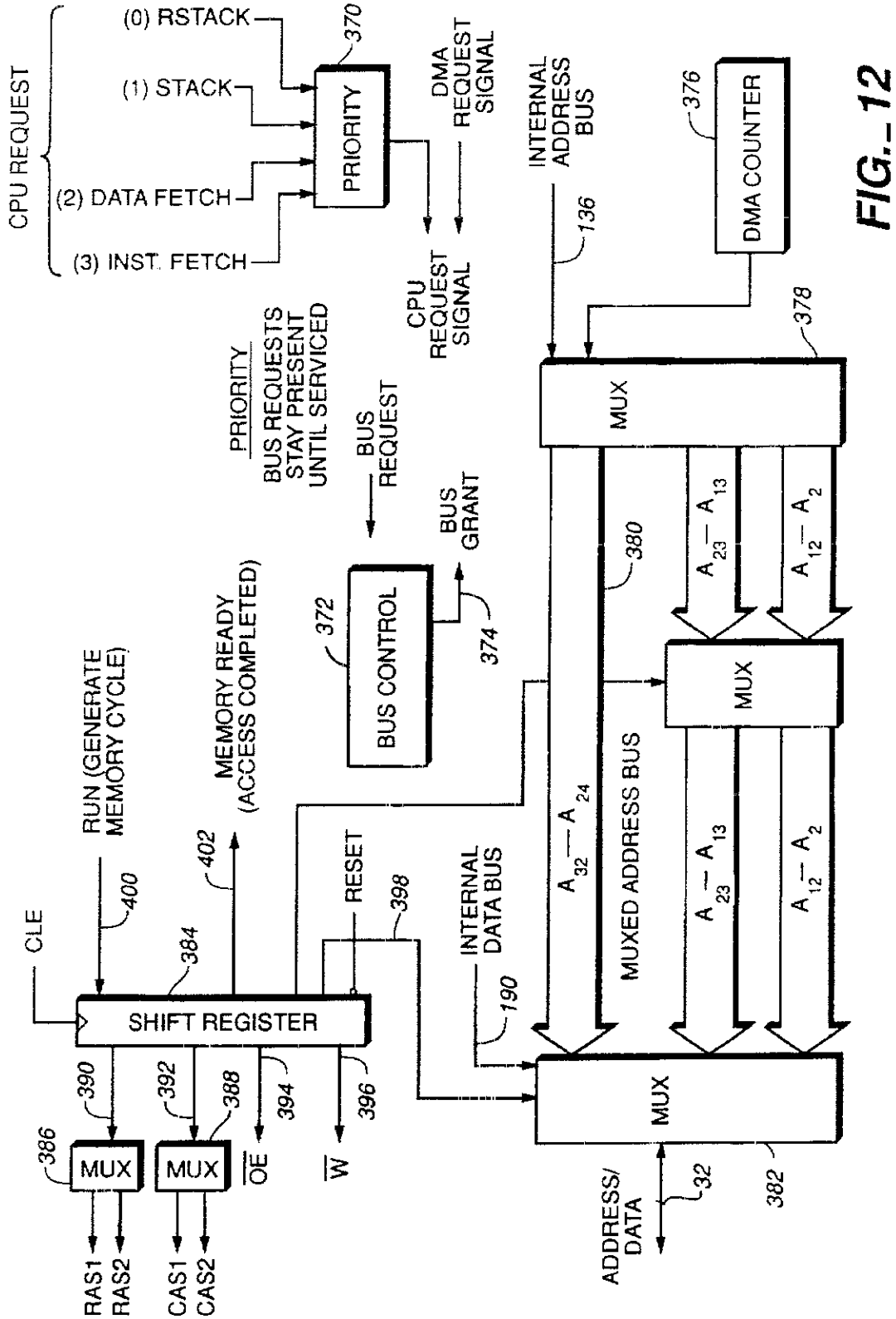

CPU REQUEST
(0) RSTACK
(1) STACK
(2) DATA FETCH
(3) INST. FETCH
370
PRIORITY
DMA REQUEST SIGNAL
CPU REQUEST SIGNAL
INTERNAL ADDRESS BUS
136
376
DMA COUNTER
378
MUX
PRIORITY
BUS REQUESTS STAY PRESENT UNTIL SERVICED
BUS REQUEST
BUS GRANT
372
BUS CONTROL
374
380
A23 — A13
A12 — A2
MUX
MEMORY READY (ACCESS COMPLETED)
RUN (GENERATE MEMORY CYCLE)
402
400
CLE
384
RESET
398
INTERNAL DATA BUS
A32 — A24
MUXED ADDRESS BUS
190
SHIFT REGISTER
390
392
388
394
396
386
MUX
MUX
OE
W
MUX
RAS1
RAS2
CAS1
CAS2
ADDRESS/ DATA
32
382

FIG._12

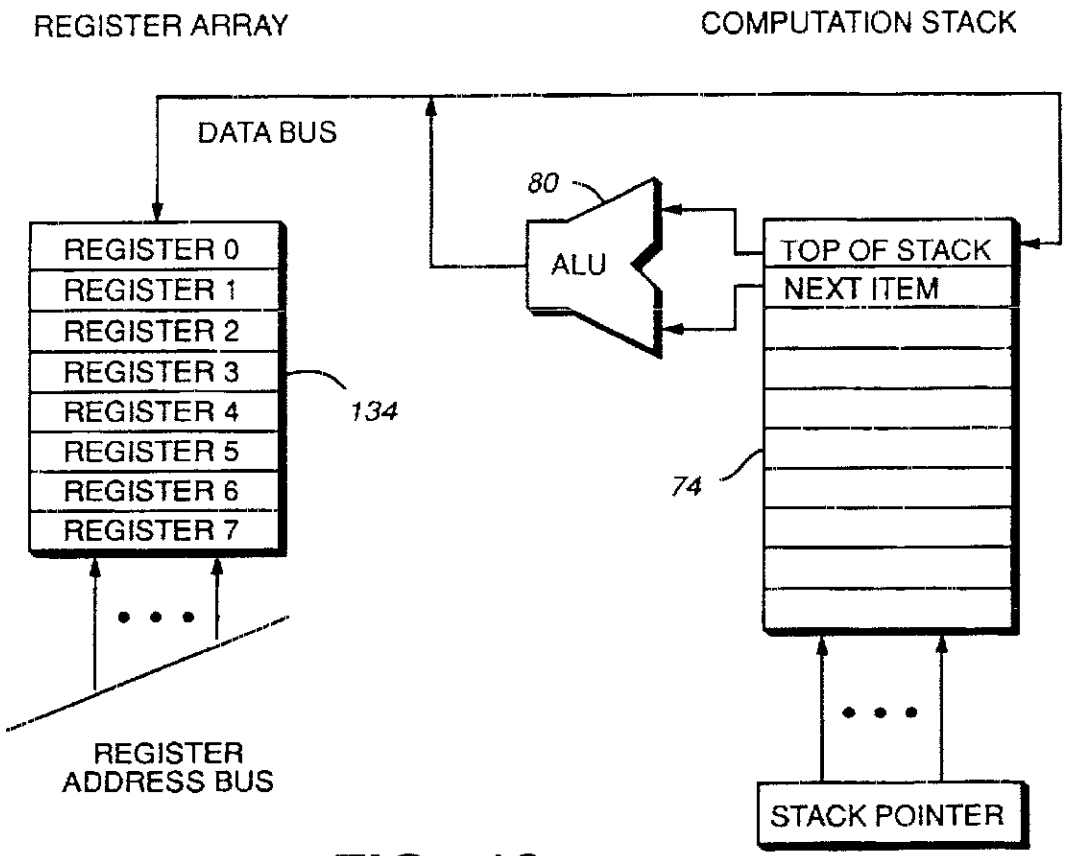

REGISTER ARRAY
COMPUTATION STACK
DATA BUS
80
ALU
REGISTER 0
REGISTER 1
REGISTER 2
REGISTER 3
REGISTER 4
REGISTER 5
REGISTER 6
REGISTER 7
134
TOP OF STACK
NEXT ITEM
74
REGISTER
ADDRESS BUS
STACK POINTER

FIG._13

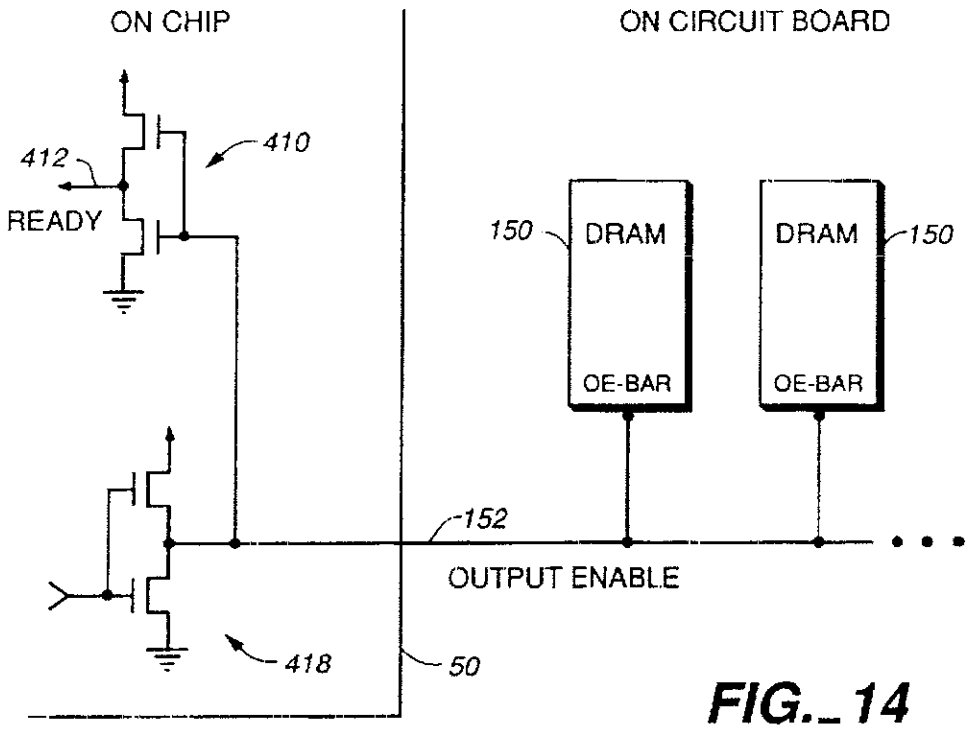

ON CHIP
ON CIRCUIT BOARD
412
410
READY
150
DRAM
OE-BAR
DRAM
150
OE-BAR
152
OUTPUT ENABLE
418
50

FIG._14

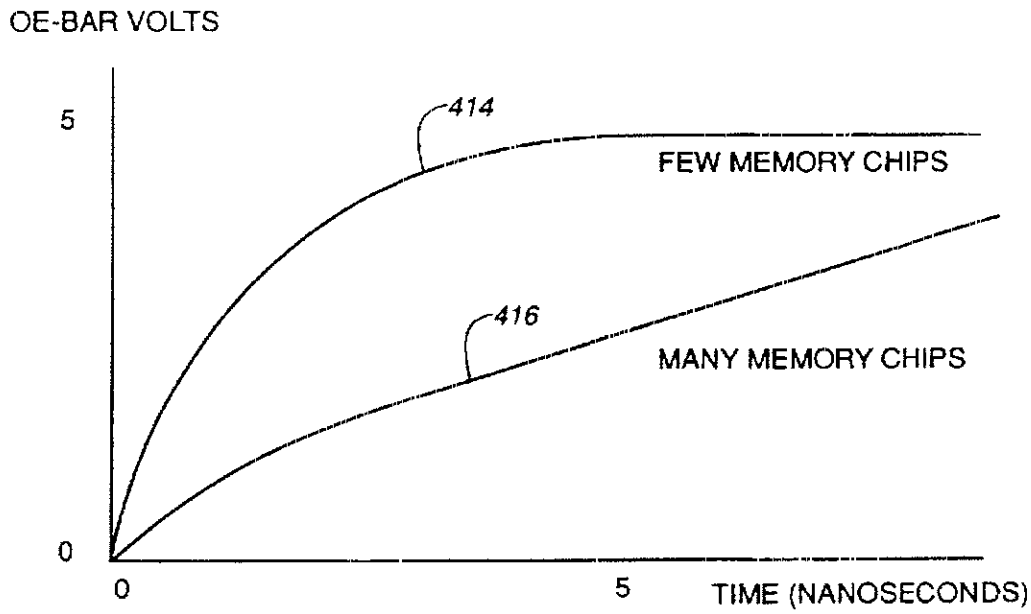

OE-BAR VOLTS
5
0
414
FEW MEMORY CHIPS
416
MANY MEMORY CHIPS
0
5
TIME (NANOSECONDS)

FIG._15

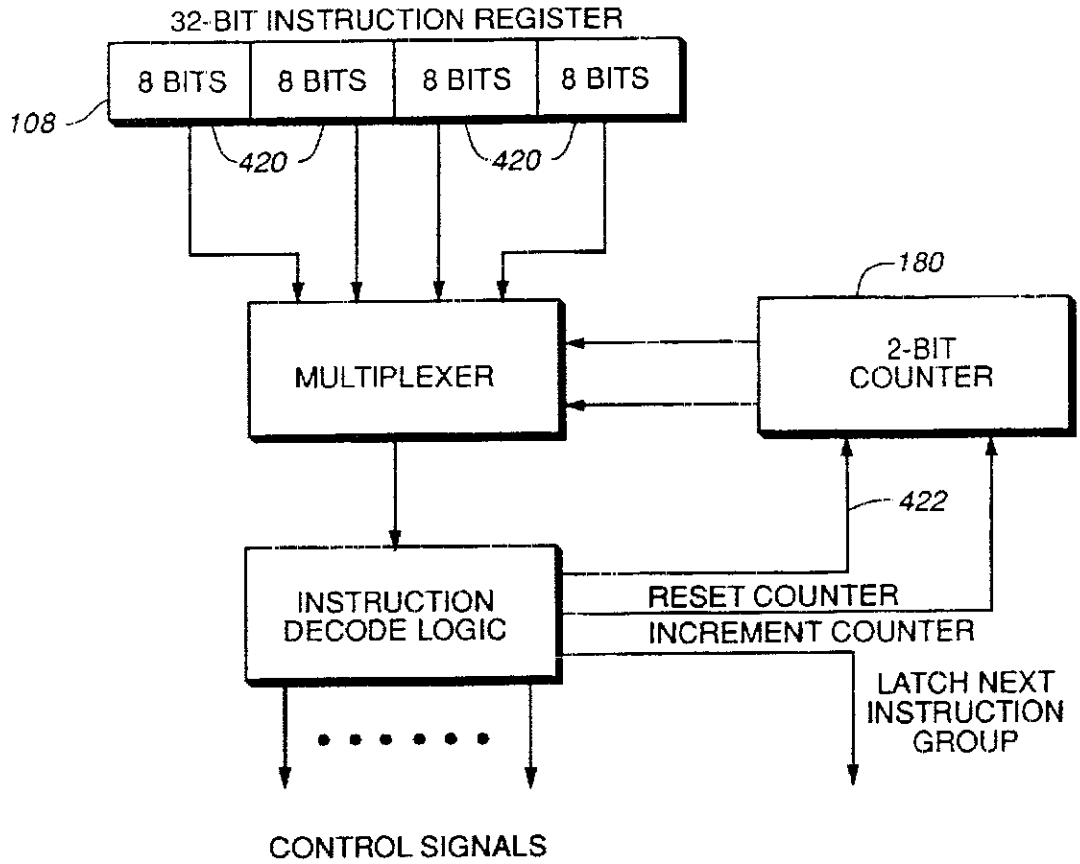

32-BIT INSTRUCTION REGISTER
8 BITS
8 BITS
8 BITS
8 BITS
108
420
420
180
MULTIPLEXER
2-BIT COUNTER
422
INSTRUCTION DECODE LOGIC
RESET COUNTER
INCREMENT COUNTER
LATCH NEXT INSTRUCTION GROUP
CONTROL SIGNALS

FIG._16

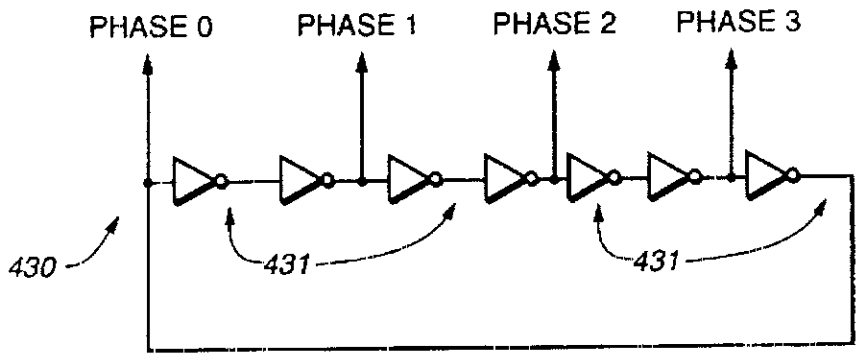

PHASE 0
PHASE 1
PHASE 2
PHASE 3
430
431
431

FIG._18

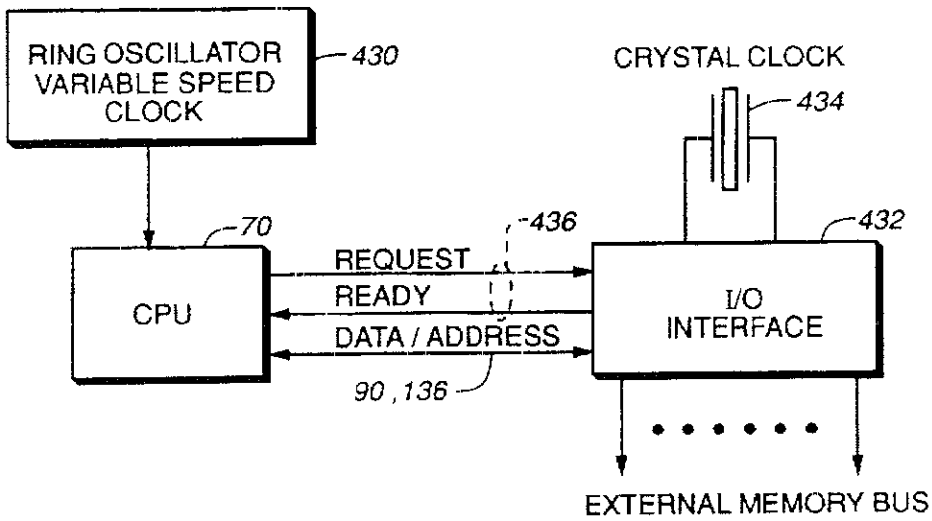

RING OSCILLATOR VARIABLE SPEED CLOCK
430
CRYSTAL CLOCK
434
70
436
432
CPU
REQUEST
READY
DATA / ADDRESS
I/O INTERFACE
90 , 136
EXTERNAL MEMORY BUS

FIG._17

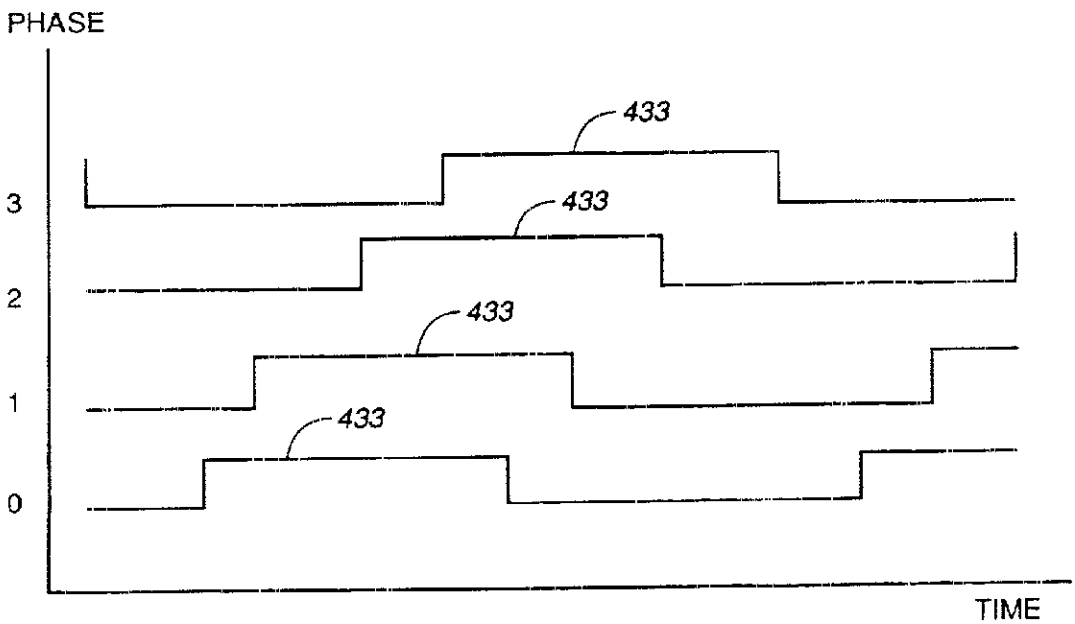

PHASE
3
2
1
0
433
433
433
433
TIME

FIG._19

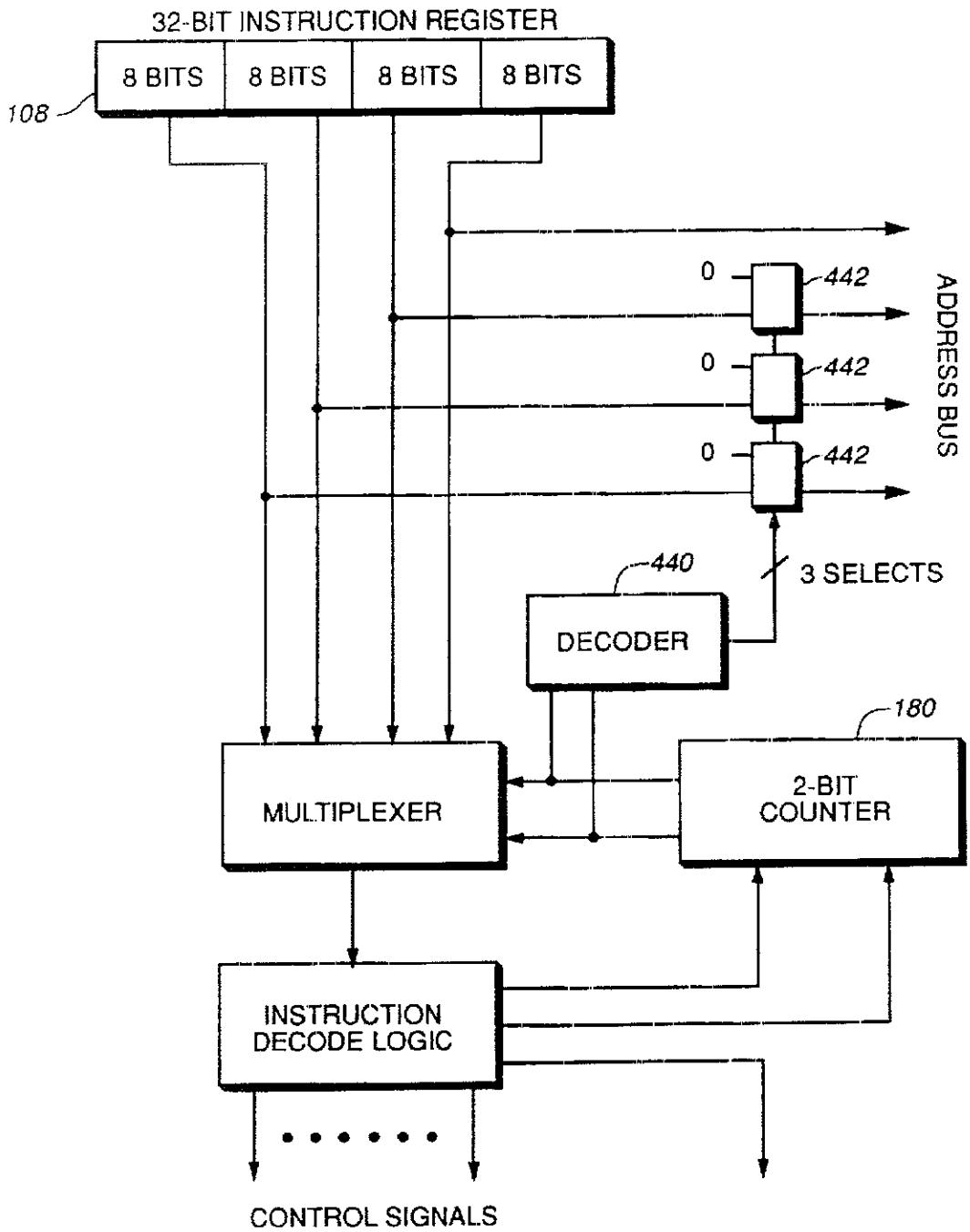

32-BIT INSTRUCTION REGISTER
8 BITS
8 BITS
8 BITS
8 BITS
108
0
442
0
442
0
442
ADDRESS BUS
440
3 SELECTS
DECODER
180
MULTIPLEXER
2-BIT COUNTER
INSTRUCTION DECODE LOGIC
CONTROL SIGNALS

FIG._20

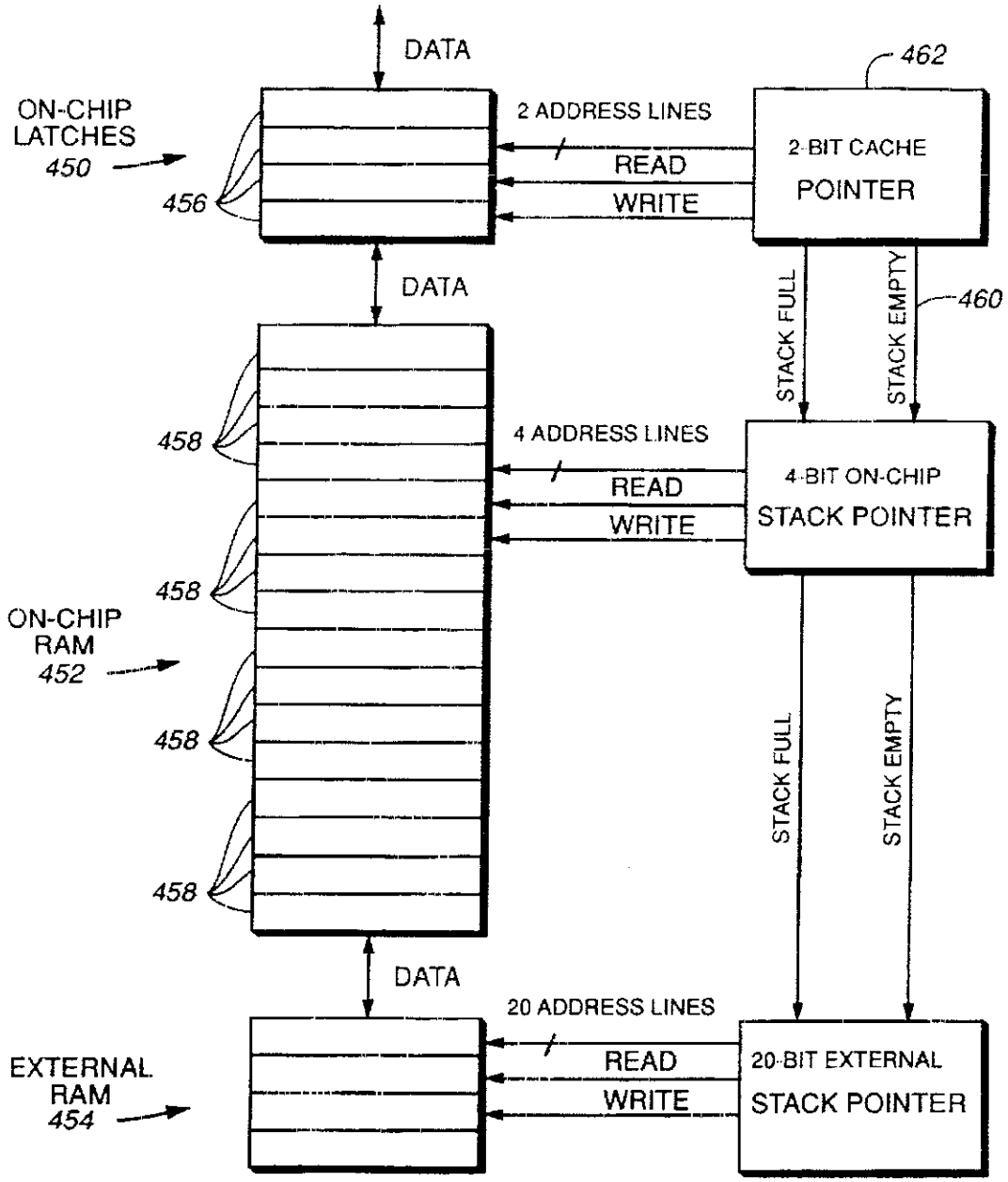

DATA
462
ON-CHIP LATCHES
450
2 ADDRESS LINES
READ
WRITE
2-BIT CACHE POINTER
456
DATA
STACK FULL
STACK EMPTY
460
458
4 ADDRESS LINES
READ
WRITE
4-BIT ON-CHIP STACK POINTER
458
ON-CHIP RAM
452
458
458
STACK FULL
STACK EMPTY
DATA
20 ADDRESS LINES
READ
WRITE
20-BIT EXTERNAL STACK POINTER
EXTERNAL RAM
454

FIG._21

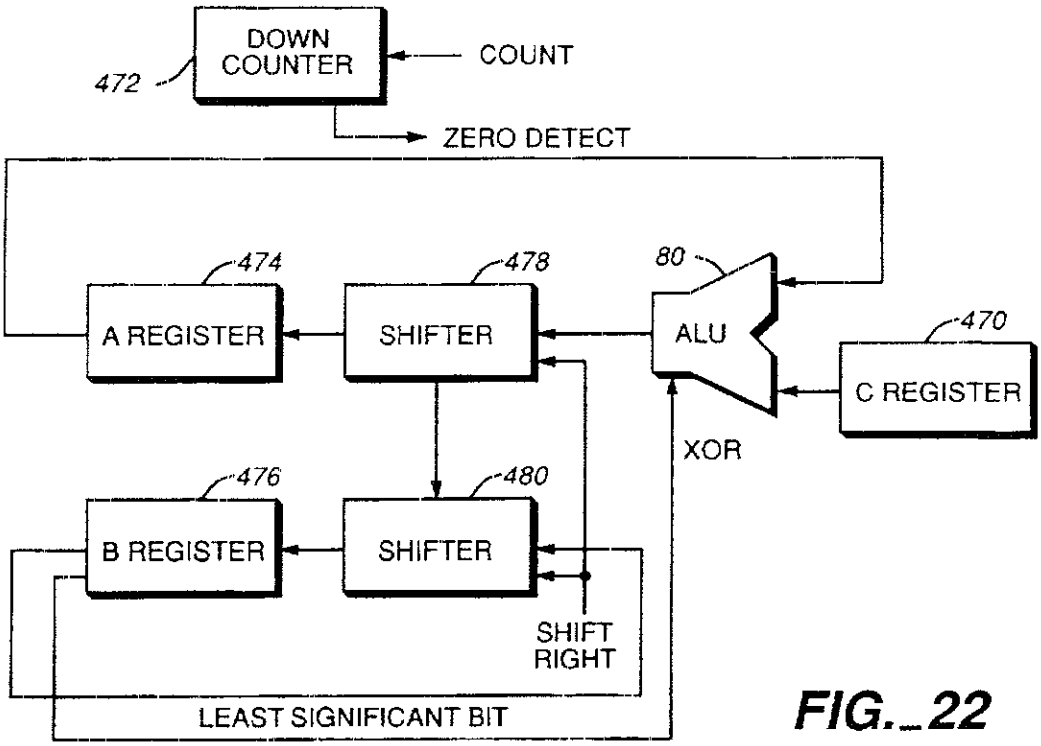

DOWN COUNTER
COUNT
472
ZERO DETECT
474
478
80
A REGISTER
SHIFTER
ALU
470
C REGISTER
XOR
476
480
B REGISTER
SHIFTER
SHIFT RIGHT
LEAST SIGNIFICANT BIT

FIG._22

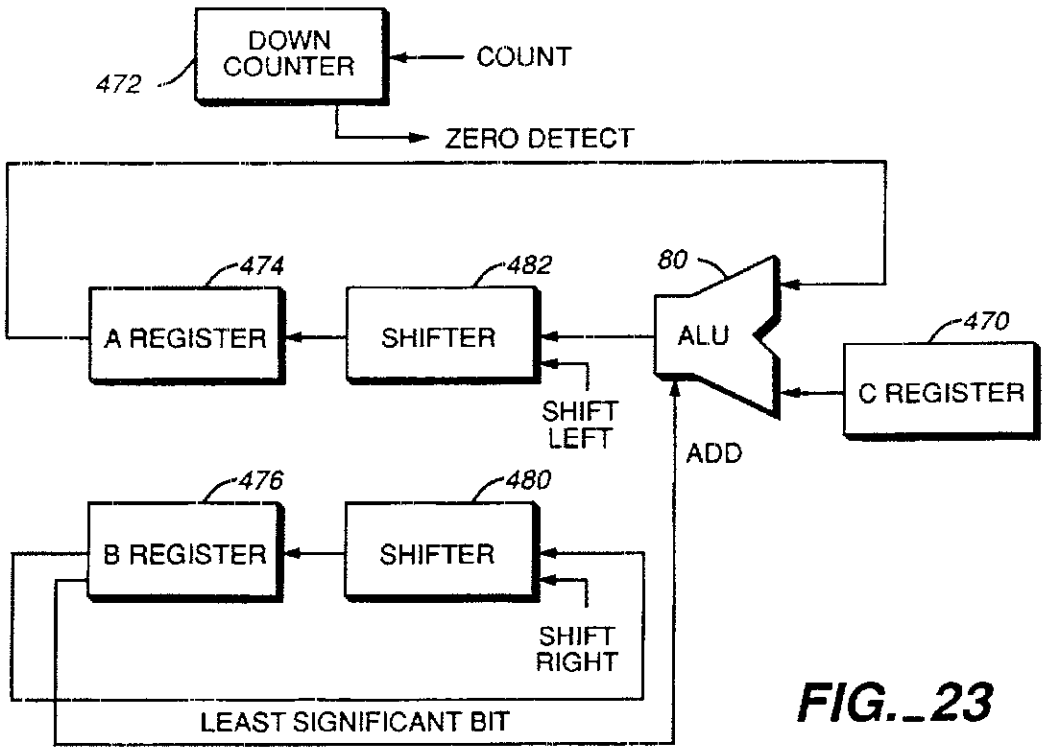

DOWN COUNTER
COUNT
472
ZERO DETECT
474
482
80
A REGISTER
SHIFTER
ALU
470
C REGISTER
SHIFT LEFT
ADD
476
480
B REGISTER
SHIFTER
SHIFT RIGHT
LEAST SIGNIFICANT BIT

FIG._23

HIGH PERFORMANCE MICROPROCESSOR USING INSTRUCTIONS THAT OPERATE WITHIN INSTRUCTION GROUPS

This application is a division of U.S. application Ser. No. 07/389,334 filed Aug. 3, 1989, now U.S. Pat. No. 5,440,749

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to a simplified, reduced instruction set computer (RISC) microprocessor. More particularly, it relates to such a microprocessor which is capable of performance levels of, for example, 20 million instructions per second (MIPS) at a price of, for example, 20 dollars.

2. Description of the Prior Art

Since the invention of the microprocessor, improvements in its design have taken two different approaches. In the first approach, a brute force gain in performance has been achieved through the provision of greater numbers of faster transistors in the microprocessor integrated circuit and an instruction set of increased complexity. This approach is exemplified by the Motorola 68000 and Intel 80X86 microprocessor families. The trend in this approach is to larger die sizes and packages with hundreds of pinouts.

More recently, it has been perceived that performance gains can be achieved through comparative simplicity, both in the microprocessor integrated circuit itself and in its instruction set. This second approach provides RISC microprocessors and is exemplified by the Sun SPARC and the Intel 8960 microprocessors. However, even with this approach as conventionally practiced, the packages for the microprocessor are large, in order to accommodate the large number of pinouts that continue to be employed. A need therefore remains for further simplification of high performance microprocessors.

With conventional high performance microprocessors, fast static memories are required for direct connection to the microprocessors in order to allow memory accesses that are fast enough to keep up with the microprocessors. Slower dynamic random access memories (DRAMs) are used with such microprocessors only in a hierarchical memory arrangement with the static memories acting as a buffer between the microprocessors and the DRAMs. The necessity to use static memories increases cost of the resulting systems.

Conventional microprocessors provide direct memory accesses (DMA) for system peripheral units through DMA controllers, which may be located on the microprocessor integrated circuit or provided separately. Such DMA controllers can provide routine handling of DMA requests and responses, but some processing by the main central processing unit (CPU) of the microprocessor is required.

SUMMARY OF THE INVENTION

Accordingly, it is an object of this invention to provide a microprocessor with a reduced pin count and cost compared to conventional microprocessors.

It is another object of the invention to provide a high performance microprocessor that can be directly connected to DRAMs without sacrificing microprocessor speed.

It is a further object of the invention to provide a high performance microprocessor in which DMA does not require use of the main CPU during DMA requests and responses and which provides very rapid DMA response with predictable response times.

The attainment of these and related objects may be achieved through use of the novel high performance, low cost microprocessor herein disclosed. In accordance with one aspect of the invention, a microprocessor system in accordance with this invention has a central processing unit, a memory and a bus connecting the central processing unit to the memory. Instruction fetching means are connected to the bus to fetch instruction groups via the bus from the memory. Each of the instruction groups include at least one instruction that accesses operands or instructions or both. The operands and instructions are located relative to the instruction groups. An instruction register receives a first of the instruction groups from the instruction fetching means. The first of the instruction groups include one or more sequential instructions. Instruction supplying means supplies, in succession from the instruction register, the one or more sequential instructions of the first of the instruction groups to the central processing unit. An instruction decoding means configures the instruction supplying means to select from the instruction register an operand associated with one of the instructions from the first of the instruction groups.

In accordance with another aspect of the invention, the microprocessor has a central processing unit and an instruction register operatively coupled to the central processing unit. An instruction fetching means provides instruction groups to the instruction register wherein certain of the instruction groups include one or more operands or sequential instructions or both. The one or more sequential instructions including at least one instruction that accesses operands or instructions or both being located relative to the instruction groups. An instruction supplying means successively couples the one or more sequential instructions of the certain of the instruction groups to the central processing unit. An instruction decoding means configures the instruction supplying means to select operands from the instruction register associated with particular ones of the sequential instructions.

In another aspect of the invention, the microprocessor system includes a central processing unit, memory, and an instruction register. A method provides instructions from the instruction register to the central processing unit and comprises the steps of:

providing instruction groups to the instruction register from the memory wherein certain of the instruction groups include one or more operands or sequential instructions or both;

supplying, in succession from the instruction register, the one or more sequential instructions of the certain of the instruction groups to the central processing unit; and

selecting an operand from the one of the instruction groups for use by the central processing unit.

The attainment of the foregoing and related objects, advantages and features of the invention should be more readily apparent to those skilled in the art after review of the following more detailed description of the invention taken together with the drawings, in which:

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is an external, plan view of an integrated circuit package incorporating a microprocessor in accordance with the invention.

FIG. 2 is a block diagram of a microprocessor in accordance with the invention.

FIG. **3** is a block diagram of a portion of a data processing system incorporating the microprocessor of FIGS. **1** and **2**.

FIG. **4** is a more detailed block diagram of a portion of the microprocessor shown in FIG. **2**.

FIG. **5** is a more detailed block diagram of another portion of the microprocessor shown in FIG. **2**.

FIG. **6** is a block diagram of another portion of the data processing system shown in part in FIG. **3** and incorporating the microprocessor of FIGS. **1–2** and **4–5**.

FIGS. **7** and **8** are layout diagrams for the data processing system shown in part in FIGS. **3** and **6**.

FIG. **9** is a layout diagram of a second embodiment of a microprocessor in accordance with the invention in a data processing system on a single integrated circuit.

FIG. **10** is a more detailed block diagram of a portion of the data processing system of FIGS. **7** and **8**.

FIG. **11** is a timing diagram useful for understanding operation of the system portion shown in FIG. **12**.

FIG. **12** is another more detailed block diagram of a further portion of the data processing system of FIGS. **7** and **8**.

FIG. **13** is a more detailed block diagram of a portion of the microprocessor shown in FIG. **2**.

FIG. **14** is a more detailed block and schematic diagram of a portion of the system shown in FIGS. **3** and **7–8**.

FIG. **15** is a graph useful for understanding operation of the system portion shown in FIG. **14**.

FIG. **16** is a more detailed block diagram showing part of the system portion shown in FIG. **4**.

FIG. **17** is a more detailed block diagram of a portion of the microprocessor shown in FIG. **2**.

FIG. **18** is a more detailed block diagram of part of the microprocessor portion shown in FIG. **17**.

FIG. **19** is a set of waveform diagrams useful for understanding operation of the part of the microprocessor portion shown in FIG. **18**.

FIG. **20** is a more detailed block diagram showing another part of the system portion shown in FIG. **4**.

FIG. **21** is a more detailed block diagram showing another part of the system portion shown in FIG. **4**.

FIGS. **22** and **23** are more detailed block diagrams showing another part of the system portion shown in FIG. **4**.

DETAILED DESCRIPTION OF THE INVENTION

OVERVIEW

The microprocessor of this invention is desirably implemented as a 32-bit microprocessor optimized for:

HIGH EXECUTION SPEED and

LOW SYSTEM COST.

In this embodiment, the microprocessor can be thought of as 20 MIPS for 20 dollars. Important distinguishing features of the microprocessor are:

Uses low-cost commodity DYNAMIC RAMS to run 20 MIPS

4 instruction fetch per memory cycle

On-chip fast page-mode memory management

Runs fast without external cache

Requires few interfacing chips

Crams 32-bit CPU in 44 pin SOJ package

The instruction set is organized so that most operations can be specified with 8-bit instructions. Two positive products of this philosophy are:

Programs are smaller.

Programs can execute much faster.

The bottleneck in most computer systems is the memory bus. The bus is used to fetch instructions and fetch and store data. The ability to fetch four instructions in a single memory bus cycle significantly increases the bus availability to handle data.

Turning now to the drawings, more particularly to FIG. **1**, there is shown a packaged 32-bit microprocessor **50** in a 44-pin plastic leadless chip carrier, shown approximately 100 times its actual size of about 0.8 inch on a side. The fact that the microprocessor **50** is provided as a 44-pin package represents a substantial departure from typical microprocessor packages, which usually have about 200 input/output (I/O) pins. The microprocessor **50** is rated at 20 million instructions per second (MIPS). Address and data lines **52**, also labelled D0–D31, are shared for addresses and data without speed penalty as a result of the manner in which the microprocessor **50** operates, as will be explained below.

DYNAMIC RAM

In addition to the low cost 44-pin package, another unusual aspect of the high performance microprocessor **50** is that it operates directly with dynamic random access memories (DRAMs) as shown by row address strobe (RAS) and column address strobe (CAS) I/O pins **54**. The other I/O pins for the microprocessor **50** include V_{DD} pins **56**, V_{SS} pins **58**, output enable pin **60**, write pin **62**, clock pin **64** and reset pin **66**.

All high speed computers require high speed and expensive memory to keep up. The highest speed static RAM memories cost as much as ten times as much as slower dynamic RAMs. This microprocessor has been optimized to use low-cost dynamic RAM in high-speed page-mode. Page-mode dynamic RAMs offer static RAM performance without the cost penalty. For example, low-cost 85 nsec dynamic RAMs access at 25 nsec when operated in fast page-mode. Integrated fast page-mode control on the microprocessor chip simplifies system interfacing and results in a faster system.

Details of the microprocessor **50** are shown in FIG. **2**. The microprocessor **50** includes a main central processing unit (CPU) **70** and a separate direct memory access (DMA) CPU **72** in a single integrated circuit making up the microprocessor **50**. The main CPU **70** has a first **16** deep push down stack **74**, which has a top item register **76** and a next item register **78**, respectively connected to provide inputs to an arithmetic logic unit (ALU) **80** by lines **82** and **84**. An output of the ALU **80** is connected to the top item register **76** by line **86**. The output of the top item register at **82** is also connected by line **88** to an internal data bus **90**.

A loop counter **92** is connected to a decrementer **94** by lines **96** and **98**. The loop counter **92** is bidirectionally connected to the internal data bus **90** by line **100**. Stack pointer **102**, return stack pointer **104**, mode register **106** and instruction register **108** are also connected to the internal data bus **90** by lines **110**, **112**, **114** and **116**, respectively. The internal data bus **90** is connected to memory controller **118** and to gate **120**. The gate **120** provides inputs on lines **122**, **124**, and **126** to X register **128**, program counter **130** and Y register **132** of return push down stack **134**. The X register **128**, program counter **130** and Y register **132** provide outputs to internal address bus **136** on lines **138**, **140** and **142**. The internal address bus provides inputs to the memory controller **118** and to an incrementer **144**. The incrementer **144** provides inputs to the X register, program counter and Y register via lines **146**, **122**, **124** and **126**. The DMA CPU **72** provides inputs to the memory controller **118** on line **148**.

The memory controller **118** is connected to a RAM (not shown) by address/data bus **151** and control lines **153**.

FIG. **2** shows that the microprocessor **50** has a simple architecture. Prior art RISC microprocessors are substantially more complex in design. For example, the SPARC RISC microprocessor has three times the gates of the microprocessor **50** and the Intel 8960 RISC microprocessor has 20 times the gates of the microprocessor **50**. The speed of this microprocessor is in substantial part due to this simplicity. The architecture incorporates push down stacks and register write to achieve this simplicity.

The microprocessor **50** incorporates an I/O that has been tuned to make heavy use of resources provided on the integrated circuit chip. On chip latches allow use of the same I/O circuits to handle three different things: column addressing, row addressing and data, with a slight to non-existent speed penalty. This triple bus multiplexing results in fewer buffers to expand, fewer interconnection lines, fewer I/O pins and fewer internal buffers.

The provision of on-chip DRAM control gives a performance equal to that obtained with the use of static RAMs. As a result, memory is provided at ¼ the system cost of static RAM used in most RISC systems.

The microprocessor **50** fetches 4 instructions per memory cycle; the instructions are in an 8-bit format, and this is a 32-bit microprocessor. System speed is therefore 4 times the memory bus bandwidth. This ability enables the microprocessor to break the Von Neumann bottleneck of the speed of getting the next instruction. This mode of operation is possible because of the use of a push down stack and register array. The push down stack allows the use of implied addresses, rather than the prior art technique of explicit addresses for two sources and a destination.

Most instructions execute in 20 nanoseconds in the microprocessor **50**. The microprocessor can therefore execute instructions at **50** peak MIPS without pipeline delays. This is a function of the small number of gates in the microprocessor **50** and the high degree of parallelism in the architecture of the microprocessor.

FIG. **3** shows how column and row addresses are multiplexed on lines D8–D14 of the microprocessor **50** for addressing DRAM **150** from I/O pins **52**. The DRAM **150** is one of eight, but only one DRAM **150** has been shown for clarity. As shown, the lines D11–D18 are respectively connected to row address inputs A0–A8 of the DRAM 15C. Additionally, lines D12–D15 are connected to the data inputs DQ1–DQ4 of the DRAM **150**. The output enable, write and column address strobe pins **54** are respectively connected to the output enable, write and column address strobe inputs of the DRAM **150** by lines **152**. The row address strobe pin **54** is connected through row address strobe decode logic **154** to the row address strobe input of the DRAM **150** by lines **156** and **158**.

D0–D7 pins **52** (FIG. **1**) are idle when the microprocessor **50** is outputting multiplexed row and column addresses on D11–D18 pins **52**. The D0–D7 pins **52** can therefore simultaneously be used for I/O when right justified I/O is desired. Simultaneous addressing and I/O can therefore be carried out.

FIG. **4** shows how the microprocessor **50** is able to achieve performance equal to the use of static RAMS with DRAMs through multiple instruction fetch in a single clock cycle and instruction fetch-ahead. Instruction register **108** receives four 8-bit byte instruction words **1–4** on 32-bit internal data bus **90**. The four instruction byte **1-4** locations of the instruction register **108** are connected to multiplexer **170** by busses **172**, **174**, **176** and **178**, respectively. A microprogram counter **180** is connected to the multiplexer **170** by lines **182**. The multiplexer **170** is connected to decoder **184** by bus **186**. The decoder **184** provides internal signals to the rest of the microprocessor **50** on lines **188**.

Most significant bits **190** of each instruction byte **1-4** location are connected to a 4-input decoder **192** by lines **194**. The output of decoder **192** is connected to memory controller **118** by line **196**. Program counter **130** is connected to memory controller **118** by internal address bus **136**, and the instruction register **108** is connected to the memory controller **118** by the internal data bus **90**. Address/data bus **198** and control bus **200** are connected to the DRAMS **150** (FIG. **3**).

In operation, when the most significant bits **190** of remaining instructions **1–4** are "1" in a clock cycle of the microprocessor **50**, there are no memory reference instructions in the queue. The output of decoder **192** on line **196** requests an instruction fetch ahead by memory controller **118** without interference with other accesses. While the current instructions in instruction register **108** are executing, the memory controller **118** obtains the address of the next set of four instructions from program counter **130** and obtains that set of instructions. By the time the current set of instructions has completed execution, the next set of instructions is ready for loading into the instruction register.

Details of the DMA CPU **72** are provided in FIG. **5**. Internal data bus **90** is connected to memory controller **118** and to DMA instruction register **210**. The DMA instruction register **210** is connected to DMA program counter **212** by bus **214**, to transfer size counter **216** by bus **218** and to timed transfer interval counter **220** by bus **222**. The DMA instruction register **210** is also connected to DMA I/O and RAM address register **224** by line **226**. The DMA I/O and RAM address register **224** is connected to the memory controller **118** by memory cycle request line **228** and bus **230**. The DMA program counter **212** is connected to the internal address bus **136** by bus **232**. The transfer size counter **216** is connected to a DMA instruction done decrementer **234** by lines **236** and **238**. The decrementer **234** receives a control input on memory cycle acknowledge line **240**. When transfer size counter **216** has completed its count, it provides a control signal to DMA program counter **212** on line **242**. Timed transfer interval counter **220** is connected to decrementer **244** by lines **246** and **248**. The decrementer **244** receives a control input from a microprocessor system clock on line **250**.

The DMA CPU **72** controls itself and has the ability to fetch and execute instructions. It operates as a co-processor to the main CPU **70** (FIG. **2**) for time specific processing.

FIG. **6** shows how the microprocessor **50** is connected to an electrically programmable read only memory (EPROM) **260** by reconfiguring the data lines **52** so that some of the data lines **52** are input lines and some of them are output lines. Data lines **52** D0–D7 provide data to and from corresponding data terminals **262** of the EPROM **260**. Data lines **52** D9–D18 provide addresses to address terminals **264** of the EPROM **260**. Data lines **52** D19–D31 provide inputs from the microprocessor **50** to memory and I/O decode logic **266**. RAS **0/1** control line **268** provides a control signal for determining whether the memory and I/O decode logic provides a DRAM RAS output on line **270** or a column enable output for the EPROM **260** on line **272**. Column address strobe terminal **60** of the microprocessor **50** provides an output enable signal on line **274** to the corresponding terminal **276** of the EPROM **260**.

FIGS **7** and **8** show the front and back of a one card data processing system **280** incorporating the microprocessor **50**. MSM514258-10 type DRAMs **150** totalling 2 megabytes, a

Motorola 50 MegaHertz crystal oscillator clock **282**. I/O circuits **284** and a 27256 type EPROM **260**. The I/O circuits **284** include a 74HC04 type high speed hex inverter circuit **286**, an IDT39C828 type 10-bit inverting buffer circuit **288**, an IDT39C822 type 10-bit inverting register circuit **290**, and two IDT39C823 type 9-bit non-inverting register circuits **292**. The card **280** is completed with a MAX12V type DC—DC converter circuit **294**, 34-pin dual AMP type headers **296**, a coaxial female power connector **298**, and a 3-pin AMP right angle header **300**. The card **280** is a low cost, imbeddable product that can be incorporated in larger systems or used as an internal development tool

The microprocessor **50** is a very high performance (50 MHz) RISC influenced 32-bit CPU designed to work closely with dynamic RAM. Clock for clock, the microprocessor **50** approaches the theoretical performance limits possible with a single CPU configuration. Eventually, the microprocessor **50** and any other processor is limited by the bus bandwidth and the number of bus paths. The critical conduit is between the CPU and memory.

One solution to the bus bandwidth/bus path problem is to integrate a CPU directly onto the memory chips, giving every memory a direct bus to the CPU. FIG. **9** shows another microprocessor **310** that is provided integrally with 1 megabit of DRAM **311** in a single integrated circuit **312**. Until the present invention, this solution has not been practical, because most high performance CPUs require from 500,000 to 1,000,000 transistors and enormous die sizes just by themselves. The microprocessor **310** is equivalent to the microprocessor **50** in FIGS. **1–8**. The microprocessors **50** and **310** are the most transistor efficient high performance CPUs in existence, requiring fewer than 50,000 transistors for dual processors **70** and **72** (FIG. **2**) or **314** and **316** (less memory). The very high speed of the microprocessors **50** and **310** is to a certain extent a function of the small number of active devices. In essence, the less silicon gets in the way, the faster the electrons can get where they are going.

The microprocessor **310** is therefore the only CPU suitable for integration on the memory chip die **312**. Some simple modifications to the basic microprocessor **50** to take advantage of the proximity to the DRAM array **311** can also increase the microprocessor **50** clock speed by 50 percent and probably more.

The microprocessor **310** core on board the DRAM die **312** provides most of the speed and functionality required for a large group of applications from automotive to peripheral control. However, the integrated CPU **310**/DRAM **311** concept has the potential to redefine significantly the way multiprocessor solutions can solve a spectrum of very compute intensive problems. The CPU **310**/DRAM **311** combination eliminates the Von Neumann bottleneck by distributing it across numerous CPU/DRAM chips **312**. The microprocessor **310** is a particularly good core for multiprocessing, since it was designed with the SDI targeting array in mind, and provisions were made for efficient interprocessor communications.

Traditional multiprocessor implementations have been very expensive in addition to being unable to exploit fully the available CPU horsepower. Multiprocessor systems have typically been built up from numerous board level or box level computers. The result is usually an immense amount of hardware with corresponding wiring, power consumption and communications problems. By the time the systems are interconnected, as much as 50 percent of the bus speed has been utilized just getting through the interfaces.

In addition, multiprocessor system software has been scarce. A multiprocessor system can easily be crippled by an inadequate load-sharing algorithm in the system software, which allows one CPU to do a great deal of work and the others to be idle. Great strides have been made recently in systems software, and even UNIX V.4 may be enhanced to support multiprocessing. Several commercial products from such manufacturers as DUAL Systems and UNISOFT do a credible job on 68030 type microprocessor systems now.

The microprocessor **310** architecture eliminates most of the interface friction, since up to **64** CPU **310**/RAM **311** processors should be able to intercommunicate without buffers or latches. Each chip **312** has about **40** MIPS raw speed, because placing the DRAM **311** next to the CPU **310** allows the microprocessor **310** instruction cycle to be cut in half, compared to the microprocessor **50**. A **64** chip array of these chips **312** is more powerful than any other existing computer. Such an array fits on a 3×5 card, cost less than a FAX machine, and draw about the same power as a small television.

Dramatic changes in price/performance always reshape existing applications and almost always create new ones. The introduction of microprocessors in the mid 1970s created video games, personal computers, automotive computers, electronically controlled appliances, and low cost computer peripherals.

The integrated circuit **312** will find applications in all of the above areas, plus create some new ones. A common generic parallel processing algorithm handles convolution/Fast Fourier Transform (FFT)/pattern recognition. Interesting product possibilities using the integrated circuit **312** include high speed reading machines, real-time speech recognition, spoken language translation, real-time robot vision, a product to identify people by their faces, and an automotive or aviation collision avoidance system.

A real time processor for enhancing high density television (HDTV) images, or compressing the HDTV information into a smaller bandwidth, would be very feasible. The load sharing in HDTV could be very straightforward. Splitting up the task according to color and frame would require 6, 9 or 12 processors. Practical implementation might require 4 meg RAMs integrated with the microprocessor **310**.

The microprocessor **310** has the following specifications

CONTROL LINES
4—POWER/GROUND
1—CLOCK
32—DATA I/O
4—SYSTEM CONTROL
EXTERNAL MEMORY FETCH
EXTERNAL MEMORY FETCH AUTOINCREMENT X
EXTERNAL MEMORY FETCH AUTOINCREMENT Y
EXTERNAL MEMORY WRITE
EXTERNAL MEMORY WRITE AUTOINCREMENT X
EXTERNAL MEMORY WRITE AUTOINCREMENT Y
EXTERNAL PROM FETCH
LOAD ALL X REGISTERS
LOAD ALL Y REGISTERS
LOAD ALL PC REGISTERS
EXCHANGE X AND Y
INSTRUCTION FETCH
ADD TO PC
ADD TO X
WRITE MAPPING REGISTER
READ MAPPING REGISTER
REGISTER CONFIGURATION

MICROPROCESSOR **310** CPU **316** CORE
COLUMN LATCH1 (1024 BITS) 32×32 MUX
STACK POINTER (16 BITS)
COLUMN LATCH2 (1024 BITS) 32×32 MUX
RSTACK POINTER (16 BITS)
PROGRAM COUNTER 32 BITS
XO REGISTER 32 BITS (ACTIVATED ONLY FOR ON-CHIP ACCESSES)
YO REGISTER 32 BITS (ACTIVATED ONLY FOR ON-CHIP ACCESSES)
LOOP COUNTER 32 BITS
DMA CPU **314** CORE
DMA PROGRAM COUNTER 24 BITS
INSTRUCTION REGISTER 32 BITS
I/O & RAM ADDRESS REGISTER 32 BITS
TRANSFER SIZE COUNTER 12 BITS
INTERVAL COUNTER 12 BITS

To offer memory expansion for the basic chip **312**, an intelligent DRAM can be produced. This chip will be optimized for high speed operation with the integrated circuit **312** by having three on-chip address registers: Program Counter, X Register and Y register. As a result, to access the intelligent DRAM, no address is required, and a total access cycle could be as short as 10 nsec. Each expansion DRAM would maintain its own copy of the three registers and would be identified by a code specifying its memory address. Incrementing and adding to the three registers will actually take place on the memory chips. A maximum of 64 intelligent DRAM peripherals would allow a large system to be created without sacrificing speed by introducing multiplexers or buffers.

There are certain differences between the microprocessor **310** and the microprocessor **50** that arise from providing the microprocessor **310** on the same die **312** with the DRAM **311**. Integrating the DRAM **311** allows architectural changes in the microprocessor **310** logic to take advantage of existing on-chip DRAM **311** circuitry. Row and column design is inherent in memory architecture. The DRAMs **311** access random bits in a memory array by first selecting a row of 1024 bits, storing them into a column latch, and then selecting one of the bits as the data to be read or written.

The time required to access the data is split between the row access and the column access. Selecting data already stored in a column latch is faster than selecting a random bit by at least a factor of six. The microprocessor **310** takes advantage of this high speed by creating a number of column latches and using them as caches and shift registers. Selecting a new row of information may be thought of as performing a 1024-bit read or write with the resulting immense bus bandwidth.

1. The microprocessor **50** treats its 32-bit instruction register **108** (see FIGS. **2** and **4**) as a cache for four 8-bit instructions. Since the DRAM **311** maintains a 1024-bit latch for the column bits, the microprocessor **310** treats the column latch as a cache for **128** 8-bit instructions. Therefore the next instruction will almost always be already present in the cache. Long loops within the cache are also possible and more useful than the 4 instruction loops in the microprocessor **50**.

2. The microprocessor **50** uses two 16×32-bit deep register arrays **74** and **134** (FIG. **2**) for the parameter stack and the return stack. The microprocessor **310** creates two other 1024-bit column latches to provide the equivalent of two 32×32-bit arrays, which can be accessed twice as fast as a register array.

3. The microprocessor **50** has a DMA capability which can be used for I/O to a video shift register. The microprocessor **310** uses yet another 1024-bit column latch as a long video shift register to drive a CRT display directly. For color displays, three on-chip shift registers could also be used. These shift registers can transfer pixels at a maximum of 100 MHz.

4. The microprocessor **50** accesses memory via an external 32-bit bus. Most of the memory **311** for the microprocessor **310** is on the same die **312**. External access to more memory is made using an 8-bit bus. The result is a smaller die, smaller package and lower power consumption than the microprocessor **50**.

5. The microprocessor **50** consumes about a third of its operating power charging and discharging the I/O pins and associated capacitances. The DRAMs **150** (FIG. **8**) connected to the microprocessor **50** dissipate most of their power in the I/O drivers. A microprocessor **310** system will consume about one-tenth the power of a microprocessor **50** system, since having the DRAM **311** next to the processor **310** eliminates most of the external capacitances to be charged and discharged.

6. Multiprocessing means splitting a computing task between numerous processors in order to speed up the solution. The popularity of multiprocessing is limited by the expense of current individual processors as well as the limited interprocessor communications ability. The microprocessor **310** is an excellent multiprocessor candidate, since the chip **312** is a monolithic computer complete with memory, rendering it low-cost and physically compact.

The shift registers implemented with the microprocessor **310** to perform video output can also be configured as interprocessor communication links. The INMOS transputer attempted a similar strategy, but at much lower speed and without the performance benefits inherent in the microprocessor **310** column latch architecture. Serial I/O is a prerequisite for many multiprocessor topologies because of the many neighbor processors which communicate. A cube has 6 neighbors. Each neighbor communicates using these lines:

DATA IN
CLOCK IN
READY FOR DATA
DATA OUT
DATA READY?
CLOCK OUT

A special start up sequence is used to initialize the on-chip DRAM **311** in each of the processors.

The microprocessor **310** column latch architecture allows neighbor processors to deliver information directly to internal registers or even instruction caches of other chips **312**. This technique is not used with existing processors, because it only improves performance in a tightly coupled DRAM system.

7. The microprocessor **50** architecture offers two types of looping structures: LOOP-IF-DONE and MICRO-LOOP. The former takes an 8-bit to 24-bit operand to describe the entry point to the loop address. The latter performs a loop entirely within the 4 instruction queue and the loop entry point is implied as the first instruction in the queue. Loops entirely within the queue run without external instruction fetches and execute up to three times as fast as the long loop construct. The microprocessor **310** retains both constructs with a few differences. The microprocessor **310** microloop functions in the same fashion as the microprocessor **50** operation, except the queue is 1024-bits or 128 8-bit instructions long. The microprocessor **310** microloop can therefore contain jumps, branches, calls and immediate operations not possible in the 4 8-bit instruction microprocessor **50** queue.

Microloops in the microprocessor **50** can only perform simple block move and compare functions. The larger microprocessor **310** queue allows entire digital signal processing or floating point algorithms to loop at high speed in the queue.

The microprocessor **50** offers four instructions to redirect execution:

CALL

BRANCH

BRANCH-IF-ZERO

LOOP-IF-NOT-DONE

These instructions take a variable length address operand 8, 16 or 24 bits long. The microprocessor **50** next address logic treats the three operands similarly by adding or subtracting them to the current program counter. For the microprocessor **310** the 16 and 24-bit operands function in the same manner as the 16 and 24-bit operands in the microprocessor **50**. The 8-bit class operands are reserved to operate entirely within the instruction queue. Next address decisions can therefore be made quickly, because only 10 bits of addresses are affected, rather than 32. There is no carry or borrow generated past the 10 bits.

8. The microprocessor **310** CPU **316** resides on an already crowded DRAM die **312**. To keep chip size as small as possible, the DMA processor **72** of the microprocessor **50** has been replaced with a more traditional DMA controller **314**. DMA is used with the microprocessor **310** to perform the following functions:

Video output to a CRT

Multiprocessor serial communications

8-bit parallel I/O

The DMA controller **314** can maintain both serial and parallel transfers simultaneously. The following DMA sources and destinations are supported by the microprocessor **310**:

DESCRIPTION	I/O	LINES
1 Video shift register	OUTPUT	1 to 3
2 Multiprocessor serial	BOTH	6 lines/channel
3 8-bit parallel	BOTH	8 data, 4 control

The three sources use separate 1024-bit buffers and separate I/O pins. Therefore all three may be active simultaneously without interference.

The microprocessor **310** can be implemented with either a single multiprocessor serial buffer or separate receive and sending buffers for each channel, allowing simultaneous bidirectional communications with six neighbors simultaneously.

FIGS. **10** and **11** provide details of the PROM DMA used in the microprocessor **50**. The microprocessor **50** executes faster than all but the fastest PROMs. PROMS are used in a microprocessor **50** system to store program segments and perhaps entire programs. The microprocessor **50** provides a feature on power-up to allow programs to be loaded from low-cost, slow speed PROMs into high speed DRAM for execution. The logic which performs this function is part of the DMA memory controller **118**. The operation is similar to DMA, but not identical, since four 8-bit bytes must be assembled on the microprocessor **50** chip, then written to the DRAM **150**.

The microprocessor **50** directly interfaces to DRAM **150** over a triple multiplexed data and address bus **350**, which carries RAS addresses, CAS addresses and data. The EPROM **260**, on the other hand, is read with non-multiplexed busses. The microprocessor **50** therefore has a special mode which unmultiplexes the data and address lines to read 8 bits of EPROM data. Four 8-bit bytes are read in this fashion. The multiplexed bus **350** is turned back on, and the data is written to the DRAM **150**.

When the microprocessor **50** detects a RESET condition, the processor stops the main CPU **70** and forces a mode 0 (PROM LOAD) instruction into the DMA CPU **72** instruction register. The DMA instruction directs the memory controller to read the EPROM **260** data at 8 times the normal access time for memory. Assuming a 50 MHz microprocessor **50**, this means an access time of 320 nsec. The instruction also indicates:

The selection address of the EPROM **260** to be loaded.

The number of 32-bit words to transfer

The DRAM **150** address to transfer into

The sequence of activities to transfer one 32-bit word from EPROM **260** to DRAM **150** are:

1. RAS goes low at 352, latching the EPROM **260** select information from the high order address bits. The EPROM **260** is selected.
2. Twelve address bits (consisting of what is normally DRAM CAS addresses plus two byte select bits are placed on the bus **350** going to the EPROM **260** address pins. These signals will remain on the lines until the data from the EPROM **260** has been read into the microprocessor **50**. For the first byte, the byte select bits will be binary 00.
3. CAS goes low at 354, enabling the EPROM **260** data onto the lower 8 bits of the external address/data bus **350**. NOTE: It is important to recognize that, during this part of the cycle, the lower 8 bits of the external data/address bus are functioning as inputs, but the rest of the bus is still acting as outputs.
4. The microprocessor **50** latches these eight least significant bits internally and shifts them 8 bits left to shift them to the next significant byte position.
5. Steps 2, 3 and 4 are repeated with byte address **01**.
6. Steps 2, 3 and 4 are repeated with byte address **10**.
7. Steps 2, 3 and 4 are repeated with byte address **11**.
8. CAS goes high at 356, taking the EPROM **260** off the data bus.
9. RAS goes high at 358, indicating the end of the EPROM **260** access.
10. RAS goes low at 360, latching the DRAM select information from the high order address bits. At the same time, the RAS address bits are latched into the DRAM **150**. The DRAM **150** is selected.
11. CAS goes low at 362, latching the DRAM **150** CAS addresses.
12. The microprocessor **50** places the previously latched EPROM **260** 32-bit data onto the external address/data bus **350**. W goes low at 364, writing the 32 bits into the DRAM **150**.
13. W goes high at 366, CAS goes high at 368. The process continues with the next word.

FIG. **12** shows details of the microprocessor **50** memory controller **118**. In operation, bus requests stay present until they are serviced. CPU **70** requests are prioritized at **370** in the order of: 1, Parameter Stack; 2, Return Stack; 3, Data Fetch; 4, Instruction Fetch. The resulting CPU request signal and a DMA request signal are supplied as bus requests to bus control **372**, which provides a bus grant signal at **374**. Internal address bus **136** and a DMA counter **376** provide inputs to a multiplexer **378**. Either a row address or a column address are provided as an output to multiplexed address bus **380** as an output from the multiplexer **378**. The multiplexed

address bus **380** and the internal data bus **90** provide address and data inputs, respectively, to multiplexer **382**. Shift register **384** supplies row address strobe (RAS) **1** and **2** control signals to multiplexer **386** and column address strobe (CAS) **1** and **2** control signals to multiplexer **388** on lines **390** and **392**. The shift register **384** also supplies output enable (OE) and write (W) signals on lines **394** and **396** and a control signal on line **398** to multiplexer **382**. The shift register **384** receives a RUN signal on line **400** to generate a memory cycle and supplies a MEMORY READY signal on line **402** when an access is complete

STACK/REGISTER ARCHITECTURE

Most microprocessors use on-chip registers for temporary storage of variables. The on-chip registers access data faster than off-chip RAM. A few microprocessors use an on-chip push down stack for temporary storage.

A stack has the advantage of faster operation compared to on-chip registers by avoiding the necessity to select source and destination registers. (A math or logic operation always uses the top two stack items as source and the top of stack as destination.) The stack's disadvantage is that it makes some operations clumsy. Some compiler activities in particular require on-chip registers for efficiency.

As shown in FIG. **13**, the microprocessor **50** provides both on-chip registers **134** and a stack **74** and reaps the benefits of both

BENEFITS:

1. Stack math and logic is twice as fast as those available on an equivalent register only machine. Most programmers and optimizing compilers can take advantage of this feature
2. Sixteen registers are available for on-chip storage of local variables which can transfer to the stack for computation. The accessing of variables is three to four times as fast as available on a strictly stack machine

The combined stack **74**/register **134** architecture has not been used previously due to inadequate understanding by computer designers of optimizing compilers and the mix of transfer versus math/logic instructions.

ADAPTIVE MEMORY CONTROLLER

A microprocessor must be designed to work with small or large memory configurations. As more memory loads are added to the data, address, and control lines, the switching speed of the signals slows down. The microprocessor **50** multiplexes the address/data bus three ways, so timing between the phases is critical. A traditional approach to the problem allocates a wide margin of time between bus phases so that systems will work with small or large numbers of memory chips connected. A speed compromise of as much as 50% is required.

As shown in FIG. **14**, the microprocessor **50** uses a feedback technique to allow the processor to adjust memory bus timing to be fast with small loads and slower with large ones. The OUTPUT ENABLE (OE) line **152** from the microprocessor **50** is connected to all memories **150** on the circuit board. The loading on the output enable line **152** to the microprocessor **50** is directly related to the number of memories **150** connected. By monitoring how rapidly OE **152** goes high after a read, the microprocessor **50** is able to determine when the data hold time has been satisfied and place the next address on the bus.

The level of the OE line **152** is monitored by CMOS input buffer **410** which generates an internal READY signal on line **412** to the microprocessor's memory controller. Curves **414** and **416** of the FIG. **15** graph show the difference in rise time likely to be encountered from a lightly to heavily loaded memory system. When the OE line **152** has reached a predetermined level to generate the READY signal, driver **418** generates an OUTPUT ENABLE signal on OE line **152**.

SKIP WITHIN THE INSTRUCTION CACHE

The microprocessor **50** fetches four 8-bit instructions each memory cycle and stores them in a 32-bit instruction register **108**, as shown in FIG. **16**. A class of "test and skip" instructions can very rapidly execute a very fast jump operation within the four instruction cache

SKIP CONDITIONS:

Always
ACC non-zero
ACC negative
Carry flag equal logic one
Never
ACC equal zero
ACC positive
Carry flag equal logic zero

The SKIP instruction can be located in any of the four byte positions **420** in the 32-bit instruction register **108**. If the test is successful, SKIP will jump over the remaining one, two, or three 8-bit instructions in the instruction register **108** and cause the next four-instruction group to be loaded into the register **108**. As shown, the SKIP operation is implemented by resetting the 2-bit microinstruction counter **180** to zero on line **422** and simultaneously latching the next instruction group into the register **108**. Any instructions following the SKIP in the instruction register are overwritten by the new instructions and not executed.

The advantage of SKIP is that optimizing compilers and smart programmers can often use it in place of the longer conditional JUMP instruction. SKIP also makes possible microloops which exit when the loop counts down or when the SKIP jumps to the next instruction group. The result is very fast code.

Other machines (such as the PDP-8 and Data General NOVA) provide the ability to skip a single instruction. The microprocessor **50** provides the ability to skip up to three instructions.

MICROLOOP IN THE INSTRUCTION CACHE

The microprocessor **50** provides the MICROLOOP instruction to execute repetitively from one to three instructions residing in the instruction register **108**. The microloop instruction works in conjunction with the LOOP COUNTER **92** (FIG. **2**) connected to the internal data bus **90**. To execute a microloop, the program stores a count in LOOP COUNTER **92**. MICROLOOP may be placed in the first, second, third, or last byte **420** of the instruction register **108**. If placed in the first position, execution will just create a delay equal to the number stored in LOOP COUNTER **92** times the machine cycle. If placed in the second, third, or last byte **420**, when the microloop instruction is executed, it will test the LOOP COUNT for zero. If zero, execution will continue with the next instruction. If not zero, the LOOP COUNTER **92** is decremented and the 2-bit microinstruction counter is cleared, causing the preceding instructions in the instruction register to be executed again.

Microloop is useful for block move and search operations. By executing a block move completely out of the instruction register **108**, the speed of the move is doubled, since all memory cycles are used by the move rather than being shared with instruction fetching. Such a hardware implementation of microloops is much faster than conventional software implementation of a comparable function.

OPTIMAL CPU CLOCK SCHEME

The designer of a high speed microprocessor must produce a product which operate over wide temperature ranges,

wide voltage swings, and wide variations in semiconductor processing. Temperature, voltage, and process all affect transistor propagation delays. Traditional CPU designs are done so that with the worse case of the three parameters, the circuit will function at the rated clock speed. The result are designs that must be clocked a factor of two slower than their maximum theoretical performance, so they will operate properly in worse case conditions.

The microprocessor **50** uses the technique shown in FIGS. **17–19** to generate the system clock and its required phases. Clock circuit **430** is the familiar "ring oscillator" used to test process performance. The clock is fabricated on the same silicon chip as the rest of the microprocessor **50**.

The ring oscillator frequency is determined by the parameters of temperature, voltage, and process. At room temperature, the frequency will be in the neighborhood of 100 MHZ. At 70 degrees Centigrade, the speed will be 50 MHZ. The ring oscillator **430** is useful as a system clock, with its stages **431** producing phase 0-phase 3 outputs **433** shown in FIG. **19**, because its performance tracks the parameters which similarly affect all other transistors on the same silicon die. By deriving system timing from the ring oscillator **430**, CPU **70** will always execute at the maximum frequency possible, but never too fast. For example, if the processing of a particular die is not good resulting in slow transistors, the latches and gates on the microprocessor **50** will operate slower than normal. Since the microprocessor **50** ring oscillator clock **430** is made from the same transistors on the same die as the latches and gates, it too will operate slower (oscillating at a lower frequency), providing compensation which allows the rest of the chip's logic to operate properly.

ASYNCHRONOUS/SYNCHRONOUS CPU

Most microprocessors derive all system timing from a single clock. The disadvantage is that different parts of the system can slow all operations. The microprocessor **50** provides a dual-clock scheme as shown in FIG. **17**, with the CPU **70** operating asynchronously to I/O interface **432** forming part of memory controller **118** (FIG. **2**) and the I/O interface **432** operating synchronously with the external world of memory and I/O devices. The CPU **70** executes at the fastest speed possible using the adaptive ring counter clock **430**. Speed may vary by a factor of four depending upon temperature, voltage, and process. The external world must be synchronized to the microprocessor **50** for operations such as video display updating and disc drive reading and writing. This synchronization is performed by the I/O interface **432**, speed of which is controlled by a conventional crystal clock **434**. The interface **432** processes requests for memory accesses from the microprocessor **50** and acknowledges the presence of I/O data. The microprocessor **50** fetches up to four instructions in a single memory cycle and can perform much useful work before requiring another memory access. By decoupling the variable speed of the CPU **70** from the fixed speed of the I/O interface **432**, optimum performance can be achieved by each. Recoupling between the CPU **70** and the interface **432** is accomplished with handshake signals on lines **436**, with data/addresses passing on bus **90**, **136**.

ASYNCHRONOUS/SYNCHRONOUS CPU IMBEDDED ON A DRAM CHIP

System performance is enhanced even more when the DRAM **311** and CPU **314** (FIG. **9**) are located on the same die. The proximity of the transistors means that DRAM **311** and CPU **314** parameters will closely follow each other. At room temperature, not only would the CPU **314** execute at 100 MHZ, but the DRAM **311** would access fast enough to keep up. The synchronization performed by the I/O interface **432** would be for DMA and reading and writing I/O ports. In some systems (such as calculators) no I/O synchronization at all would be required, and the I/O clock would be tied to the ring counter clock.

VARIABLE WIDTH OPERANDS

Many microprocessors provide variable width operands. The microprocessor **50** handles operands of 8, 16, or 24 bits using the same op-code. FIG. **20** shows the 32-bit instruction register **108** and the 2-bit microinstruction register **180** which selects the 8-bit instruction. Two classes of microprocessor **50** instructions can be greater than 8-bits, JUMP class and IMMEDIATE. A JUMP or IMMEDIATE op-code is 8-bits, but the operand can be 8, 16, or 24 bits long. This magic is possible because operands must be right justified in the instruction register. This means that the least significant bit of the operand is always located in the least significant bit of the instruction register. The microinstruction counter **180** selects which 8-bit instruction to execute. If a JUMP or IMMEDIATE instruction is decoded, the state of the 2-bit microinstruction counter selects the required 8, 16, or 24 bit operand onto the address or data bus. The unselected 8-bit bytes are loaded with zeros by operation of decoder **440** and gates **442**. The advantage of this technique is the saving of a number of op-codes required to specify the different operand sizes in other microprocessors.

TRIPLE STACK CACHE

Computer performance is directly related to the system memory bandwidth. The faster the memories, the faster the computer. Fast memories are expensive, so techniques have been developed to move a small amount of high-speed memory around to the memory addresses where it is needed. A large amount of slow memory is constantly updated by the fast memory, giving the appearance of a large fast memory array. A common implementation of the technique is known as a high-speed memory cache. The cache may be thought of as fast acting shock absorber smoothing out the bumps in memory access. When more memory is required than the shock can absorb, it bottoms out and slow speed memory is accessed. Most memory operations can be handled by the shock absorber itself.

The microprocessor **50** architecture has the ALU **80** (FIG **2**) directly coupled to the top two stack locations **76** and **78**. The access time of the stack **74** therefore directly affects the execution speed of the processor. The microprocessor **50** stack architecture is particularly suitable to a triple cache technique, shown in FIG. **21** which offers the appearance of a large stack memory operating at the speed of on-chip latches **450**. Latches **450** are the fastest form of memory device built on the chip, delivering data in as little as 3 nsec. However latches **450** require large numbers of transistors to construct. On-chip RAM **452** requires fewer transistors than latches, but is slower by a factor of five (15 nsec access). Off-chip RAM **150** is the slowest storage of all. The microprocessor **50** organizes the stack memory hierarchy as three interconnected stacks **450**, **452** and **454**. The latch stack **450** is the fastest and most frequently used. The on-chip RAM stack **452** is next. The off-chip RAM stack **454** is slowest. The stack modulation determines the effective access time of the stack. If a group of stack operations never push or pull more than four consecutive items on the stack, operations will be entirely performed in the 3 nsec latch stack. When the four latches **456** are filled, the data in the bottom of the latch stack **450** is written to the top of the on-chip RAM stack **452**. When the sixteen locations **458** in the on-chip RAM stack **452** are filled, the data in the bottom of the on-chip RAM stack **452** is written to the top of the off-chip

RAM stack **454**. When popping data off a full stack **450**, four pops will be performed before stack empty line **460** from the latch stack pointer **462** transfers data from the on-chip RAM stack **452**. By waiting for the latch stack **450** to empty before performing the slower on-chip RAM access, the high effective speed of the latches **456** are made available to the processor. The same approach is employed with the on-chip RAM stack **452** and the off-chip RAM stack **454**.

POLYNOMIAL GENERATION INSTRUCTION

Polynomials are useful for error correction, encryption, data compression, and fractal generation. A polynomial is generated by a sequence of shift and exclusive OR operations. Special chips are provided for this purpose in the prior art.

The microprocessor **50** is able to generate polynomials at high speed without external hardware by slightly modifying how the ALU **80** works. As shown in FIG. **22**, a polynomial is generated by loading the 'order" (also known as the feedback terms) into C Register **470**. The value thirty one (resulting in 32 iterations) is loaded into DOWN COUNTER **472**. A register **474** is loaded with zero. B register **476** is loaded with the starting polynomial value. When the POLY instruction executes, C register **470** is exclusively ORed with A register **474** if the least significant bit of B register **476** is a one. Otherwise, the contents of the A register **474** passes through the ALU **80** unaltered. The combination of A and B is then shifted right (divided by 2) with shifters **478** and **480**. The operation automatically repeats the specified number of iterations, and the resulting polynomial is left in A register **474**.

FAST MULTIPLY

Most microprocessors offer a 16×16 or 32×32 bit multiply instruction. Multiply when performed sequentially takes one shift/add per bit, or 32 cycles for 32 bit data. The microprocessor **50** provides a high speed multiply which allows multiplication by small numbers using only a small number of cycles. FIG. **23** shows the logic used to implement the high speed algorithm. To perform a multiply, the size of the multiplier less one is placed in the DOWN COUNTER **472**. For a four bit multiplier, the number three would be stored in the DOWN COUNTER **472**. Zero is loaded into the A register **474**. The multiplier is written bit reversed into the B Register **476**. For example, a bit reversed five (binary 0101) would be written into B as 1010. The multiplicand is written into the C register **470**. Executing the FAST MULT instruction will leave the result in the A Register **474** when the count has been completed. The fast multiply instruction is important because many applications scale one number by a much smaller number. The difference in speed between multiplying a 32×32 bit and a 32×4 bit is a factor of 8. If the least significant bit of the multiplier is a "ONE", the contents of the A register **474** and the C register **470** are added. If the least significant bit of the multiplier is a "ZERO", the contents of the A register are passed through the ALU **80** unaltered. The output of the ALU **80** is shifted left by shifter **482** in each iteration. The contents of the B register **476** are shifted right by the shifter **480** in each iteration.

INSTRUCTION EXECUTION PHILOSOPHY

The microprocessor **50** uses high speed D latches in most of the speed critical areas. Slower on-chip RAM is used as secondary storage.

The microprocessor **50** philosophy of instruction execution is to create a hierarchy of speed as follows:

Logic and D latch transfers	1 cycle	20 nsec
Math	2 cycles	40 nsec
Fetch/store on-chip RAM	2 cycles	40 nsec
Fetch/store in current RAS page	4 cycles	80 nsec
Fetch/store with RAS cycle	11 cycles	220 nsec

With a 50 MHZ clock many operations can be performed in 20 nsec, and almost everything else in 40 nsec.

To maximize speed, certain techniques in processor design have been used. They include:

Eliminating arithmetic operations on addresses.

Fetching up to four instructions per memory cycle.

Pipelineless instruction decoding.

Generating results before they are needed.

Use of three level stack caching.

PIPELINE PHILOSOPHY

Computer instructions are usually broken down into sequential pieces, for example: fetch, decode, register read, execute, and store. Each piece will require a single machine cycle. In most Reduced Instruction Set Computer (RISC) chips, instruction require from three to six cycles.

RISC instructions are very parallel. For example, each of 70 different instructions in the SPARC (SUN Computer's RISC chip) has five cycles. Using a technique called "pipelining", the different phases of consecutive instructions can be overlapped.

To understand pipelining, think of building five residential homes. Each home will require in sequence, a foundation, framing, plumbing and wiring, roofing, and interior finish. Assume that each activity takes one week. To build one house will take five weeks.

But what if you want to build an entire subdivision? You have only one of each work crew, but when the foundation men finish on the first house, you immediately start them on the second one, and so on. At the end of five weeks, the first home is complete, but you also have five foundations. If you have kept the framing, plumbing, roofing, and interior guys all busy, from five weeks on, a new house will be completed each week.

This is the way a RISC chip like SPARC appears to execute an instruction in a single machine cycle. In reality, a RISC chip is executing one fifth of five instructions each machine cycle. And if five instructions stay in sequence, an instruction will be completed each machine cycle.

The problems with a pipeline are keeping the pipe full with instructions. Each time an out of sequence instruction such as a BRANCH or CALL occurs, the pipe must be refilled with the next sequence. The resulting dead time to refill the pipeline can become substantial when many IF/THEN/ELSE statements or subroutines are encountered.

THE PIPELINE APPROACH

The microprocessor **50** has no pipeline as such. The approach of this microprocessor to speed is to overlap instruction fetching with execution of the previously fetched instruction(s). Beyond that, over half the instructions (the most common ones) execute entirely in a single machine cycle of 20 nsec. This is possible because:

1. Instruction decoding resolves in 2.5 nsec.
2. Incremented/decremented and some math values are calculated before they are needed, requiring only a latching signal to execute.
3. Slower memory is hidden from high speed operations by high-speed D latches which access in 4 nsec.

The disadvantage for this microprocessor is a more complex chip design process. The advantage for the chip user is faster

ultimate throughput since pipeline stalls cannot exist. Pipeline synchronization with availability flag bits and other such pipeline handling is not required by this microprocessor.

For example in some RISC machines an instruction which tests a status flag may have to wait for up to four cycles for the flag set by the previous instruction to be available to be tested. Hardware and software debugging is also somewhat easier because the user doesn't have to visualize five instructions simultaneously in the pipe.

OVERLAPPING INSTRUCTION FETCH/EXECUTE

The slowest procedure the microprocessor **50** performs is to access memory. Memory is accessed when data is read or written. Memory is also read when instructions are fetched. The microprocessor **50** is able to hide fetch of the next instruction behind the execution of the previously fetched instruction(s). The microprocessor **50** fetches instructions in 4-byte instruction groups. An instruction group may contain from one to four instructions. The amount of time required to execute the instruction group ranges from 4 cycles for simple instructions to 64 cycles for a multiply.

When a new instruction group is fetched, the microprocessor instruction decoder looks at the most significant bit of all four of the bytes. The most significant bit of an instruction determines if a memory access is required. For example, CALL, FETCH, and STORE all require a memory access to execute. If all four bytes have nonzero most significant bits, the microprocessor initiates the memory fetch of the next sequential 4-byte instruction group. When the last instruction in the group finishes executing, the next 4-byte instruction group is ready and waiting on the data bus needing only to be latched into the instruction register. If the 4-byte instruction group required four or more cycles to execute and the next sequential access was a column address strobe (CAS) cycle, the instruction fetch was completely overlapped with execution.

INTERNAL ARCHITECTURE

The microprocessor **50** architecture consists of the following:

PARAMETER STACK <--> ALU* <--> Y REGISTER RETURN STACK	
<---32 BITS---> 16 DEEP	<---32 BITS---> 16 DEEP
Used for math and logic	Used for subroutine and interrupt return addresses as well as local variables
Push down stack. Can overflow into off-chip RAM.	Push down stack. Can overflow into off-chip RAM. Can also be accessed relative to top of stack.
LOOP COUNTER	(32-bits, can decrement by 1) Used by class of test and loop instructions.
X REGISTER	(32-bits, can increment or decrement by 4). Used to point to RAM locations
PROGRAM COUNTER	(32-bits, increments by 4). Points to 4-byte instruction groups in RAM
INSTRUCTION REG	(32-Bits). Holds 4-byte instruction groups while they are being decoded and executed

* Math and logic operations use the TOP item and NEXT to top Parameter Stack items as the operands. The result is pushed onto the Parameter Stack.

* Return addresses from subroutines are placed on the Return Stack. The Y REGISTER is used as a pointer to RAM locations. Since the Y REGISTER is the top item of the Return Stack, nesting of indices is straightforward.

MODE—A register with mode and status bits

MODE-BITS:

Slow down memory accesses by 8 if "1". Run full speed if "0". (Provided for access to slow EPROM.)

Divide the system clock by 1023 if "1" to reduce power consumption. Run full speed if "0". (On-chip counters slow down if this bit is set.)

Enable external interrupt **1**

Enable external interrupt **2**

Enable external interrupt **3**

Enable external interrupt **4**

Enable external interrupt **5**

Enable external interrupt **6**

Enable external interrupt **7**.

ON-CHIP MEMORY LOCATIONS:

MODE-BITS

DMA-POINTER

DMA-COUNTER

STACK-POINTER—Pointer into Parameter Stack.

STACK-DEPTH—Depth of on-chip Parameter Stack

RSTACK-POINTER—Pointer into Return Stack

RSTACK-DEPTH—Depth of on-chip Return Stack

ADDRESSING MODE HIGH POINTS

The data bus is 32-bits wide. All memory fetches and stores are 32-bits. Memory bus addresses are 30 bits. The least significant 2 bits are used to select one-of-four bytes in some addressing modes. The Program Counter, X Register, and Y Register are implemented as D latches with their outputs going to the memory address bus and the bus incrementer/decrementer. Incrementing one of these registers can happen quickly, because the incremented value has already rippled through the inc/dec logic and need only be clocked into the latch. Branches and Calls are made to 32-bit word boundaries.

INSTRUCTION SET

32-BIT INSTRUCTION FORMAT

The thirty two bit instructions are CALL, BRANCH, BRANCH-IF-ZERO, and LOOP-IF-NOT-DONE. These instructions require the calculation of an effective address. In many computers, the effective address is calculated by adding or subtracting an operand with the current Program Counter. This math operation requires from four to seven machine cycles to perform and can definitely bog down machine execution. The microprocessor's strategy is to perform the required math operation at assembly or linking time and do a much simpler "Increment to next page" or "Decrement to previous page" operation at run time. As a result, the microprocessor branches execute in a single cycle.

24-BIT OPERAND FORM:

Byte **1** Byte **2** Byte **3** Byte **4**

WWWWWW XX—YYYYYYYY—YYYYYYYY—YYYYYYYY

With a 24-bit operand, the current page is considered to be defined by the most significant 6 bits of the Program Counter.

16-BIT OPERAND FORM:

QQQQQQQQ—WWWWWW XX—YYYYYYYY—YYYYYYYY

With a 16-bit operand, the current page is considered to be defined by the most significant 14 bits of the Program Counter

8-BIT OPERAND FORM:

QQQQQQQQ—QQQQQQQQ—WWWWWW XX—YYYYYYYY

With an 8-bit operand the current page is considered to be defined by the most significant 22 bits of the Program Counter.

QQQQQQQQ—Any 8-bit instruction

WWWWWW—Instruction op-code.

XX—Select how the address bits will be used:

00—Make all high-order bits zero. (Page zero addressing)

01—Increment the high-order bits. (Use next page)

10—Decrement the high-order bits. (Use previous page)

11—Leave the high-order bits unchanged. (Use current page)

YYYYYYYY—The address operand field. This field is always shifted left two bits (to generate a word rather than byte address) and loaded into the Program Counter. The microprocessor instruction decoder figures out the width of the operand field by the location of the instruction op-code in the four bytes.

The compiler or assembler will normally use the shortest operand required to reach the desired address so that the leading bytes can be used to hold other instructions. The effective address is calculated by combining:

The current Program Counter.

The 8, 16, or 24 bit address operand in the instruction.

Using one of the four allowed addressing modes.

EXAMPLES OF EFFECTIVE ADDRESS CALCULATION

EXAMPLE 1 :

Byte 1	Byte 2	Byte 3	Byte 4
QQQQQQQQ	QQQQQQQQ	00000011	10011000

The "QQQQQQQQs" in Byte **1** and **2** indicate space in the 4-byte memory fetch which could be hold two other instructions to be executed prior to the CALL instruction. Byte **3** indicates a CALL instruction (six zeros) in the current page (indicated by the 11 bits). Byte **4** indicates that the hexadecimal number 98 will be forced into the Program Counter bits **2** through **10**. (Remember, a CALL or BRANCH always goes to a word boundary so the two least significant bits are always set to zero). The effect of this instruction would be to CALL a subroutine at WORD location HEX 98 in the current page. The most significant 22 bits of the Program Counter define the current page and will be unchanged.

EXAMPLE 2 :

Byte 1	Byte 2	Byte 3	Byte 4
000001 01	00000001	00000000	00000000

If we assume that the Program Counter was HEX 0000 0156 which is binary:

00000000 00000000 00000001 01010110=OLD PROGRAM COUNTER.

Byte **1** indicates a BRANCH instruction op code (000001) and "01" indicates select the next page. Byte **2**,**3**, and **4** are the address operand. These 24-bits will be shifted to the left two places to define a WORD address. HEX 0156 shifted left two places is HEX 0558. Since this is a 24-bit operand instruction, the most significant 6 bits of the Program Counter define the current page. These six bits will be incremented to select the next page. Executing this instruction will cause the Program Counter to be loaded with HEX 0400 0558 which is binary:

00000100 00000000 00000101 01011000 = NEW PROGRAM COUNTER.

INSTRUCTIONS

CALL-LONG

0000 00XX – YYYYYYYY – YYYYYYYY – YYYYYYYY

Load the Program Counter with the effective WORD address specified. Push the current PC contents onto the RETURN STACK.

OTHER EFFECTS: CARRY or modes, no effect. May cause Return Stack to force an external memory cycle if on-chip Return Stack is full.

BRANCH

0000 01XX—YYYYYYYY—YYYYYYYY—YYYYYYYY

Load the Program Counter with the effective WORD address specified.

OTHER EFFECTS: NONE

BRANCH-IF-ZERO

0000 10XX—YYYYYYYY—YYYYYYYY—YYYYYYYY

Test the TOP value on the Parameter Stack. If the value is equal to zero, load the Program Counter with the effective WORD address specified. If the TOP value is not equal to zero, increment the Program Counter and fetch and execute the next instruction.

OTHER EFFECTS: NONE

LOOP-IF-NOT-DONE

0000 11YY—(XXXX XXXX)—(XXXX XXXX)—(XXXX XXXX)

If the LOOP COUNTER is not zero, load the Program Counter with the effective WORD address specified. If the LOOP COUNTER is zero, decrement the LOOP COUNTER, increment the Program Counter and fetch and execute the next instruction.

OTHER EFFECTS: NONE

8-BIT INSTRUCTIONS PHILOSOPHY

Most of the work in the microprocessor **50** is done by the 8-bit instructions. Eight bit instructions are possible with the microprocessor because of the extensive use of implied stack addressing. Many 32-bit architectures use 8-bits to specify the operation to perform but use an additional 24-bits to specify two sources and a destination.

For math and logic operations, the microprocessor **50** exploits the inherent advantage of a stack by designating the source operand(s) as the top stack item and the next stack item. The math or logic operation is performed, the operands are popped from the stack, and the result is pushed back on the stack. The result is a very efficient utilization of instruction bits as well as registers. A comparable situation exists between Hewlett Packard calculators (which use a stack) and Texas Instrument calculators which don't. The identical operation on an HP will require one half to one third the keystrokes of the TI.

The availability of 8-bit instructions also allows another architectural innovation, the fetching of four instructions in a single 32-bit memory cycle. The advantages of fetching multiple instructions are:

Increased execution speed even with slow memories.

Similar performance to the Harvard (separate data and instruction busses) without the expense.

Opportunities to optimize groups of instructions.

The capability to perform loops within this mini-cache.

The microloops inside the four instruction group are effective for searches and block moves

SKIP INSTRUCTIONS

The microprocessor **50** fetches instructions in 32-bit chunks called 4-byte instruction groups. These four bytes may contain four 8-bit instructions or some mix of 8-bit and 16 or 24-bit instructions. SKIP instructions in the microprocessor skip any remaining instructions in a 4-byte instruction group and cause a memory fetch to get the next 4-byte instruction group. Conditional SKIPs when combined with 3-byte BRANCHES will create conditional BRANCHES. SKIPs may also be used in situations when no use can be made of the remaining bytes in a 4-instruction group. A SKIP executes in a single cycle, whereas a group of three NOPs would take three cycles.

SKIP-ALWAYS—Skip any remaining instructions in this 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group.

SKIP-IF-ZERO—If the TOP item of the Parameter Stack is zero skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the TOP item is not zero, execute the next sequential instruction.

SKIP-IF-POSITIVE—If the TOP item of the Parameter Stack has a the most significant bit (the sign bit) equal to "0", skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the TOP item is not "0", execute the next sequential instruction.

SKIP-IF-NO-CARRY—If the CARRY flag from a SHIFT or arithmetic operation is not equal to "1", skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the CARRY is equal to "1", execute the next sequential instruction.

SKIP-NEVER Execute the next sequential (NOP) instruction. (Delay one machine cycle).

SKIP-IF-NOT-ZERO—If the TOP item on the Parameter Stack is not equal to "0", skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the TOP item is equal "0", execute the next sequential instruction.

SKIP-IF-NEGATIVE—If the TOP item on the Parameter Stack has its most significant bit (sign bit) set to "1" skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the TOP item has its most significant bit set to "0", execute the next sequential instruction.

SKIP-IF-CARRY—If the CARRY flag is set to "1" as a result of SHIFT or arithmetic operation, skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the CARRY flag is "0", execute the next sequential instruction

MICROLOOPS

Microloops are a unique feature of the microprocessor architecture which allows controlled looping within a 4-byte instruction group. A microloop instruction tests the LOOP COUNTER for "0" and may perform an additional test. If the LOOP COUNTER is not "0" and the test is met, instruction execution continues with the first instruction in the 4-byte instruction group, and the LOOP COUNTER is decremented. A microloop instruction will usually be the last byte in a 4-byte instruction group, but it can be any byte. If the LOOP COUNTER is "0" or the test is not met, instruction execution continues with the next instruction. If the microloop is the last byte in the 4-byte instruction group, the most significant 30-bits of the Program Counter are incremented and the next 4-byte instruction group is fetched from memory. On a termination of the loop on LOOP COUNTER equal to "0" the LOOP COUNTER will remain at "0". Microloops allow short iterative work such as moves and searches to be performed without slowing down to fetch instructions from memory.

EXAMPLE:

Byte 1	Byte 2
FETCH-VIA-X-AUTOINCREMENT	STORE-VIA-Y-AUTOINCREMENT
Byte 3	**Byte 4**
ULOOP-UNTIL-DONE	QQQQQQQQ

This example will perform a block move. To initiate the transfer, X will be loaded with the starting address of the source. Y will be loaded with the starting address of the destination. The LOOP COUNTER will be loaded with the number of 32-bit words to move. The microloop will FETCH and STORE and count down the LOOP COUNTER until it reaches zero. QQQQQQQQ indicates any instruction can follow.

MICROLOOP INSTRUCTIONS

ULOOP-UNTIL-DONE—If the LOOP COUNTER is not "0", continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0", continue execution with the next instruction.

ULOOP-IF-ZERO—If the LOOP COUNTER is not "0" and the TOP item on the Parameter Stack is "0", continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the TOP item is "1", continue execution with the next instruction

ULOOP-IF-POSITIVE—If the LOOP COUNTER is not "0" and the most significant bit (sign bit) is "0", continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the TOP item is "1", continue execution with the next instruction.

ULOOP-IF-NOT-CARRY-CLEAR—If the LOOP COUNTER is not "0" and the floating point exponents found in TOP and NEXT are not aligned, continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the exponents are aligned, continue execution with the next instruction. This instruction is specifically designed for combination with special SHIFT instructions to align two floating point numbers.

ULOOP-NEVER—(DECREMENT-LOOP-COUNTER) Decrement the LOOP COUNTER. Continue execution with the next instruction.

ULOOP-IF-NOT-ZERO—If the LOOP COUNTER is not "0" and the TOP item of the Parameter Stack is "0", continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the TOP item is "1", continue execution with the next instruction.

ULOOP-IF-NEGATIVE—If the LOOP COUNTER is not "0" and the most significant bit (sign bit) of the TOP item of the Parameter Stack is "1" continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the most significant bit of the Parameter Stack is "0", continue execution with the next instruction

ULOOP-IF-CARRY-SET—If the LOOP COUNTER is not "0" and the exponents of the floating point numbers found in TOP and NEXT are not aligned, continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the exponents are aligned, continue execution with the next instruction.

RETURN FROM SUBROUTINE OR INTERRUPT

Subroutine calls and interrupt acknowledgements cause a redirection of normal program execution. In both cases, the current Program Counter is pushed onto the Return Stack so the microprocessor can return to its place in the program after executing the subroutine or interrupt service routine.

NOTE: When a CALL to subroutine or interrupt is acknowledged the Program Counter has already been incremented and is pointing to the 4-byte instruction group following the 4-byte group currently being executed. The instruction decoding logic allows the microprocessor to perform a test and execute a return conditional on the outcome of the test in a single cycle. A RETURN pops an address from the Return Stack and stores it to the Program Counter.

RETURN INSTRUCTIONS

RETURN-ALWAYS—Pop the top item from the Return Stack and transfer it to the Program Counter.

RETURN-IF-ZERO—If the TOP item on the Parameter Stack is "0", pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

RETURN-IF-POSITIVE—If the most significant bit (sign bit) of the TOP item on the Parameter Stack is a "0", pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction

RETURN-IF-CARRY-CLEAR—If the exponents of the floating point numbers found in TOP and NEXT are not aligned, pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

RETURN-NEVER—Execute the next instruction (NOP)

RETURN-IF-NOT-ZERO—If the TOP item on the Parameter Stack is not "0", pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

RETURN-IF-NEGATIVE—If the most significant bit (sign bit) of the TOP item on the Parameter Stack is a "1", pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

RETURN-IF-CARRY-SET—If the exponents of the floating point numbers found in TOP and NEXT are aligned, pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.

HANDLING MEMORY FROM DYNAMIC RAM

The microprocessor **50**, like any RISC type architecture is optimized to handle as many operations as possible on-chip for maximum speed. External memory operations take from 80 nsec. to 220 nsec. compared with on-chip memory speeds of from 4 nsec. to 30 nsec. There are times when external memory must be accessed.

External memory is accessed using three registers:

X-REGISTER—A 30-bit memory pointer which can be used for memory access and simultaneously incremented or decremented

Y-REGISTER—A 30-bit memory pointer which can be used for memory access and simultaneously incremented or decremented

PROGRAM-COUNTER—A 30-bit memory pointer normally used to point to 4-byte instruction groups. External memory may be accessed at addresses relative to the PC. The operands are sometimes called "Immediate" or "Literal" in other computers. When used as memory pointer, the PC is also incremented after each operation.

MEMORY LOAD & STORE INSTRUCTIONS

FETCH-VIA-X—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. X is unchanged.

FETCH-VIA-Y—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. Y is unchanged.

FETCH-VIA-X-AUTOINCREMENT—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. After fetching, increment the most significant 30 bits of X to point to the next 32-bit word address.

FETCH-VIA-Y-AUTOINCREMENT—Fetch the 32-bit memory content pointed to by Y and push it onto the Parameter Stack. After fetching, increment the most significant 30 bits of Y to point to the next 32-bit word address.

FETCH-VIA-X-AUTODECREMENT—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. After fetching, decrement the most significant 30 bits of X to point to the previous 32-bit word address.

FETCH-VIA-Y-AUTODECREMENT—Fetch the 32-bit memory content pointed to by Y and push it onto the Parameter Stack. After fetching, decrement the most significant 30 bits of Y to point to the previous 32-bit word address.

STORE-VIA-X—Pop the top item of the Parameter Stack and store it in the memory location pointed to by X. X is unchanged.

STORE-VIA-Y—Pop the top item of the Parameter Stack and store it in the memory location pointed to by Y. Y is unchanged.

STORE-VIA-X-AUTOINCREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by X. After storing, increment the most significant 30 bits of X to point to the next 32-bit word address.

STORE-VIA-Y-AUTOINCREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by Y. After storing, increment the most significant 30 bits of Y to point to the next 32-bit word address.

STORE-VIA-X-AUTODECREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by X. After storing, decrement the most significant 30 bits of X to point to the previous 32-bit word address.

STORE-VIA-Y-AUTODECREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by Y. After storing, decrement the most significant 30 bits of Y to point to the previous 32-bit word address.

FETCH-VIA-PC—Fetch the 32-bit memory content pointed to by the Program Counter and push it onto the Parameter

Stack. After fetching increment the most significant 30 bits of the Program Counter to point to the next 32-bit word address.

*NOTE When this instruction executes, the PC is pointing to the memory location following the instruction The effect is of loading a 32-bit immediate operand. This is an 8-bit instruction and therefore will be combined with other 8-bit instructions in a 4-byte instruction fetch It is possible to have from one to four FETCH-VIA-PC instructions in a 4-byte instruction fetch. The PC increments after each execution of FETCH-VIA-PC, so it is possible to push four immediate operands on the stack The four operands would be the found in the four memory locations following the instruction.

BYTE-FETCH-VIA-X—Fetch the 32-bit memory content pointed to by the most significant 30 bits of X. Using the two least significant bits of X, select one of four bytes from the 32-bit memory fetch right justify the byte in a 32-bit field and push the selected byte preceded by leading zeros onto the Parameter Stack

BYTE-STORE-VIA-X—Fetch the 32-bit memory content pointed to by the most significant 30 bits of X. Pop the TOP item from the Parameter Stack.

Using the two least significant bits of X place the least significant byte into the 32-bit memory data and write the 32-bit entity back to the location pointed to by the most significant 30 bits of X.

OTHER EFFECTS OF MEMORY ACCESS INSTRUCTIONS:

Any FETCH instruction will push a value on the Parameter Stack **74**. If the on-chip stack is full, the stack will overflow into off-chip memory stack resulting in an additional memory cycle. Any STORE instruction will pop a value from the Parameter Stack **74**. If the on-chip stack is empty a memory cycle will be generated to fetch a value from off-chip memory stack.

HANDLING ON-CHIP VARIABLES

High-level languages often allow the creation of LOCAL VARIABLES These variables are used by a particular procedure and discarded. In cases of nested procedures layers of these variables must be maintained On-chip storage is up to five times faster than off-chip RAM, so a means of keeping local variables on-chip can make operations run faster. The microprocessor **50** provides the capability for both on-chip storage of local variables and nesting of multiple levels of variables through the Return Stack.

The Return Stack **134** is implemented as 16 on-chip RAM locations. The most common use for the Return Stack **134** is storage of return addresses from subroutines and interrupt calls. The microprocessor allows these 16 locations to also be used as addressable registers. The 16 locations may be read and written by two instructions which indicate a Return Stack relative address from 0–15. When high-level procedures are nested, the current procedure variables push the previous procedure variables further down the Return Stack **134**. Eventually, the Return Stack will automatically overflow into off-chip RAM.

ON-CHIP VARIABLE INSTRUCTIONS

READ-LOCAL-VARIABLE XXXX—Read the XXXXth location relative to the top of the Return Stack. (XXXX is a binary number from 0000–1111). Push the item read onto the Parameter Stack.

OTHER EFFECTS: If the Parameter Stack is full, the push operation will cause a memory cycle to be generated as one item of the stack is automatically stored to external RAM. The logic which selects the location performs a modulo 16 subtraction If four local variables have been pushed onto the Return Stack, and an instruction attempts to READ the fifth item, unknown data will be returned

WRITE-LOCAL-VARIABLE XXXX—Pop the TOP item of the Parameter Stack and write it into the XXXXth location relative to the top of the Return Stack. (XXXX is a binary number from 0000–1111)

OTHER EFFECTS: If the Parameter Stack is empty the pop operation will cause a memory cycle to be generated to fetch the Parameter Stack item from external RAM. The logic which selects the location performs a modulo 16 subtraction If four local variables have been pushed onto the Return Stack, and an instruction attempts to WRITE to the fifth item, it is possible to clobber return addresses or wreak other havoc.

REGISTER AND FLIP-FLOP TRANSFER AND PUSH INSTRUCTIONS

DROP—Pop the TOP item from the Parameter Stack and discard it.

SWAP—Exchange the data in the TOP Parameter Stack location with the data in the NEXT Parameter Stack location

DUP—Duplicate the TOP item on the Parameter Stack and push it onto the Parameter Stack

PUSH-LOOP-COUNTER—Push the value in LOOP COUNTER onto the Parameter Stack.

POP-RSTACK-PUSH-TO-STACK—Pop the top item from the Return Stack and push it onto the Parameter Stack

PUSH-X-REG—Push the value in the X Register onto the Parameter Stack

PUSH-STACK-POINTER—Push the value of the Parameter Stack pointer onto the Parameter Stack.

PUSH-RSTACK-POINTER—Push the value of the Return Stack pointer onto the Return Stack

PUSH-MODE-BITS—Push the value of the MODE REGISTER onto the Parameter Stack

PUSH-INPUT—Read the 10 dedicated input bits and push the value (right justified and padded with leading zeros) onto the Parameter Stack

SET-LOOP-COUNTER—Pop the TOP value from the Parameter Stack and store it into LOOP COUNTER

POP-STACK-PUSH-TO-RSTACK—Pop the TOP item from the Parameter Stack and push it onto the Return Stack

SET-X-REG—Pop the TOP item from the Parameter Stack and store it into the X Register

SET-STACK-POINTER—Pop the TOP item from the Parameter Stack and store it into the Stack Pointer.

SET-RSTACK-POINTER—Pop the TOP item from the Parameter Stack and store it into the Return Stack Pointer

SET-MODE-BITS—Pop the TOP value from the Parameter Stack and store it into the MODE BITS

SET-OUTPUT—Pop the TOP item from the Parameter Stack and output it to the 10 dedicated output bits.

OTHER EFFECTS: Instructions which push or pop the Parameter Stack or Return Stack may cause a memory cycle as the stacks overflow back and forth between on-chip and off-chip memory

LOADING A SHORT LITERAL

A special case of register transfer instruction is used to push an 8-bit literal onto the Parameter Stack. This instruction requires that the 8-bits to be pushed reside in the last byte of a 4-byte instruction group. The instruction op-code loading the literal may reside in ANY of the other three bytes in the instruction group.

EXAMPLE:

BYTE 1	BYTE 2	BYTE 3
LOAD-SHORT-LITERAL	QQQQQQQQ	QQQQQQQQ
BYTE 4		
00001111		

In this example, QQQQQQQQ indicates any other 8-bit instruction. When Byte 1 is executed, binary 00001111 (HEX 0f) from Byte 4 will be pushed (right justified and padded by leading zeros) onto the Parameter Stack. Then the instructions in Byte 2 and Byte 3 will execute. The microprocessor instruction decoder knows not to execute Byte 4. It is possible to push three identical 8-bit values as follows:

BYTE 1	BYTE 2
LOAD-SHORT-LITERAL	LOAD-SHORT-LITERAL
BYTE 3	BYTE 4
LOAD-SHORT-LITERAL	00001111

SHORT-LITERAL-INSTRUCTION

LOAD-SHORT-LITERAL—Push the 8-bit value found in Byte 4 of the current 4-byte instruction group onto the Parameter Stack.

LOGIC INSTRUCTIONS

Logical and math operations use the stack for the source of one or two operands and as the destination for results. The stack organization is a particularly convenient arrangement for evaluating expressions. TOP indicates the top value on the Parameter Stack 74. NEXT indicates the next to top value on the Parameter Stack 74.

AND—Pop TOP and NEXT from the Parameter Stack, perform the logical AND operation on these two operands, and push the result onto the Parameter Stack.

OR—Pop TOP and NEXT from the Parameter Stack, perform the logical OR operation on these two operands, and push the result onto the Parameter Stack.

XOR—Pop TOP and NEXT from the Parameter Stack, perform the logical exclusive OR on these two operands, and push the result onto the Parameter Stack.

BIT-CLEAR—Pop TOP and NEXT from the Parameter Stack, toggle all bits in NEXT, perform the logical AND operation on TOP, and push the result onto the Parameter Stack. (Another way of understanding this instruction is thinking of it as clearing all bits in TOP that are set in NEXT.)

MATH INSTRUCTIONS

Math instruction pop the TOP item and NEXT to top item of the Parameter Stack 74 to use as the operands. The results are pushed back on the Parameter Stack. The CARRY flag is used to latch the "33rd bit" of the ALU result.

ADD—Pop the TOP item and NEXT to top item from the Parameter Stack, add the values together and push the result back on the Parameter Stack. The CARRY flag may be changed.

ADD-WITH-CARRY—Pop the TOP item and the NEXT to top item from the Parameter Stack, add the values together. If the CARRY flag is "1" increment the result. Push the ultimate result back on the Parameter Stack. The CARRY flag may be changed.

ADD-X—Pop the TOP item from the Parameter Stack and read the third item from the top of the Parameter Stack. Add the values together and push the result back on the Parameter Stack. The CARRY flag may be changed.

SUB—Pop the TOP item and NEXT to top item from the Parameter Stack. Subtract NEXT from TOP and push the result back on the Parameter Stack. The CARRY flag may be changed.

SUB-WITH-CARRY—Pop the TOP item and NEXT to top item from the Parameter Stack. Subtract NEXT from TOP. If the CARRY flag is "1" increment the result. Push the ultimate result back on the Parameter Stack. The CARRY flag may be changed.

SUB-X—

SIGNED-MULT-STEP—

UNSIGNED-MULT-STEP SIGNED-FAST-MULT—

FAST-MULT-STEP—

UNSIGNED-DIV-STEP—

GENERATE-POLYNOMIAL—

ROUND—

COMPARE—Pop the TOP item and NEXT to top item from the Parameter Stack. Subtract NEXT from TOP. If the result has the most significant bit equal to "0" (the result is positive), push the result onto the Parameter Stack. If the result has the most significant bit equal to "1" (the result is negative), push the old value of TOP onto the Parameter Stack. The CARRY flag may be affected.

SHIFT/ROTATE

SHIFT-LEFT—Shift the TOP Parameter Stack item left one bit. The CARRY flag is shifted into the least significant bit of TOP.

SHIFT-RIGHT—Shift the TOP Parameter Stack item right one bit. The least significant bit of TOP is shifted into the CARRY flag. Zero is shifted into the most significant bit of TOP.

DOUBLE-SHIFT-LEFT—Treating the TOP item of the Parameter Stack as the most significant word of a 64-bit number and the NEXT stack item as the least significant word, shift the combined 64-bit entity left one bit. The CARRY flag is shifted into the least significant bit of NEXT.

DOUBLE-SHIFT-RIGHT—Treating the TOP item of the Parameter Stack as the most significant word of a 64-bit number and the NEXT stack item as the least significant word, shift the combined 64-bit entity right one bit. The least significant bit of NEXT is shifted into the CARRY flag. Zero is shifted into the most significant bit of TOP.

OTHER INSTRUCTIONS

FLUSH-STACK—Empty all on-chip Parameter Stack locations into off-chip RAM. (This instruction is useful for multitasking applications) This instruction accesses a counter which holds the depth of the on-chip stack and can require from none to 16 external memory cycles.

FLUSH-RSTACK—Empty all on-chip Return Stack locations into off-chip RAM. (This instruction is useful for multitasking applications) This instruction accesses a counter which holds the depth of the on-chip Return Stack and can require from none to 16 external memory cycles.

It should further be apparent to those skilled in the art that various changes in form and details of the invention as shown and described may be made. It is intended that such changes be included within the spirit and scope of the claims appended hereto.

What is claimed is:

1. A microprocessor system comprising:

a central processing unit;

memory;

a bus connecting said central processing unit to said memory;

instruction fetching means that are connected to said bus to fetch instruction groups via said bus from said memory, certain of said instruction groups including at least one instruction that, when executed, causes an access to an operand or an instruction or both, said operand or instruction being located a predetermined position from a boundary of said instruction groups;

an instruction register for receiving sequential instructions from a first of said instruction groups from said instruction fetching means, said first of said instruction groups including said at least one instruction;

instruction decoding means having means for generating a counter control signal and an operand control signal;

a counter that is connected to receive said counter control signal from said instruction decoding means;

operand selection means that is responsive to said operand control signal from said instruction decoding means;

instruction supplying means, responsive to said counter to select said predetermined position, for supplying, in succession from said instruction register, said sequential instructions to said central processing unit;

said instruction supplying means being further responsive to said counter and said operand selection means for selecting and supplying operand from said predetermined position in said instruction groups to said central processing unit;

said instruction decoding means providing said counter control signal and said operand control signal to cause said instruction supplying means to select from said instruction groups said operand or instruction or both associated with one of said instructions from said first of said instruction groups.

2. The microprocessor system of claim **1** wherein said instruction decoding means further includes means, responsive to a SKIP instruction in said instruction register, for configuring said instruction fetching means such that the next instruction group is supplied to the instruction register, and for configuring said instruction supplying means to supply in succession from said instruction register said sequential instructions, beginning with the first instruction in said instruction register from said next instruction group, to said central processing unit, and in which said means for generating counter control signal also in response to the SKIP instruction supplies the counter control signal to reset said counter to zero.

3. The microprocessor system of claim **2** further comprising:

means for determining whether a predefined condition exists within said microprocessor system, and

means for controlling response of said instruction decoding means to said SKIP instruction and said predefined condition to execute or not execute said SKIP instruction based on existence of said predefined condition.

4. The microprocessor system of claim **1** further comprising:

a loop counter that is connected to receive a decrement control signal from said instruction decoding means, said instruction decoding means further including means, responsive to a MICROLOOP instruction in said instruction register, configured to supply said decrement control signal to said loop counter, said instruction supplying means being configured to supply from said instruction register beginning with the first instruction in said instruction register from said first of said instruction groups, to said central processing unit, and in which said means for generating the counter control signal, also in response to the MICROLOOP instruction, supplies the counter control signal for resetting said counter to zero.

5. The microprocessor system of claim **4** further comprising:

means for determining whether a predefined condition exists within said microprocessor system, and

means for controlling response of said instruction decoding means to said MICROLOOP instruction and said predefined condition to execute or not execute said MICROLOOP instruction based on existence of said predefined condition.

6. The microprocessor system of claim **1** wherein said instruction decoding means includes means for supplying control signals to said instruction fetching means such that a subsequent one of said instruction groups is supplied to said instruction register, and for configuring said instruction supplying means to supply to said central processing unit a remainder of said first of said instruction groups as said operand.

7. The microprocessor system of claim **6** wherein said instruction decoding means are configured to supply control signals to said instruction fetching means such that a subsequent one of said instruction groups supplied to said instruction register is determined in response to a branch-type instruction in said sequential instructions within said first of said instruction groups.

8. The microprocessor system of claim **1** wherein said instruction decoding means configures said instruction supplying means to supply to said central processing unit a last byte of said first of said instruction groups as said operand in response to one of said sequential instructions within said first of said instruction groups.

9. The microprocessor system of claim **1** wherein said instruction decoding means are configured to supply control signals to said instruction fetching means such that a subsequent one of said instruction groups is supplied as an operand in response to one of said sequential instructions within said first of said instruction groups.

10. The microprocessor system of claim **1** wherein said instruction decoding means are configured to supply control signals to said instruction fetching means such that a subsequent one of said instruction groups supplied to said instruction register is determined in response to a branch-type instruction in said sequential instructions within said first of said instruction groups.

11. The microprocessor system of claim **10** in which said instruction decoding means supplies said counter control signal to reset said counter in response to a branch-type instruction in said sequential instructions within said first of said instruction groups.

12. The microprocessor system of claim **10** further comprising means for determining whether a predefined condition exists within said microprocessor system, and

means for controlling response of said instruction decoding means to said branch-type instruction and said predefined condition to execute or not execute said branch-type instruction based on existence of said predefined condition.

13. The microprocessor system of claim **10** in which said instruction supplying means includes means for gating said sequential instructions within said instruction register to said central processing unit based on signals produced by said counter.

14. The microprocessor system of claim **1** wherein said instruction fetching means fetches said sequential instructions in parallel for each of said instruction groups in a single memory cycle.

15. The microprocessor system of claim **1** further comprising:

memory access testing means for testing said first of said instruction groups to determine if said sequential instructions require a memory access; and

if said memory access testing means determine a memory access is not required, then supplying of control signals

to said instruction fetching means to fetch the next instruction group during the execution of a current of said instruction groups.

16. The microprocessor of claim 1 wherein said instruction supplying means includes:

a decoder connected to an output of said counter, and

a plurality of gates interposed between said instruction register and said central processing unit said gates being controlled by signals from said decoder

17. The microprocessor of claim 1 wherein said instruction decoding means includes means for determining a width of said operand, said width being related to position in said instruction register of said one of said instructions of said first of said instruction groups.

18. The microprocessor of claim 1 wherein said first of said instruction groups includes a first instruction and multiple operand bytes, said instruction decoding means including means for determining a width of said operand associated with said first instruction based on position of said first instruction within said instruction register.

19. The microprocessor of claim 18 wherein said instruction supplying means includes gating means for selecting one or more of said multiple operand bytes within said instruction register corresponding to said operand.

20. A microprocessor comprising;

a central processing unit;

an instruction register operatively coupled to said central processing unit;

instruction fetching means for providing sequential instructions within instruction groups to said instruction register wherein certain of said instruction groups include at least one instruction that, when executed, causes an access to an operand or an instruction or both said operand or instruction being located at a predetermined position from a boundary of said instruction groups;

instruction decoding means having a means for generating a counter control signal and an operand control signal;

a counter that is connected to receive said counter control signal from said instruction decoding means;

operand selection means that is responsive to said operand control signal from said instruction decoding means;

instruction supplying means, responsive to said counter to select said predetermined position, for successively coupling said sequential instructions of said certain of said instruction groups to said central processing unit;

said instruction supplying means being further responsive to said counter and said operand selection means for selection and supplying operands from said predetermined position in said instruction groups to said central processing unit; and

said instruction decoding means providing said counter control signal and said operand control signal to cause said instruction supplying means to select from said instruction groups said operand or instruction or both associated with particular ones of said sequential instructions

21. The microprocessor of claim 20 wherein said instruction decoding means, upon receiving a SKIP one of said sequential instructions from a current one of said instruction groups configures said instruction fetching means to fetch a next one of said instruction groups to said instruction register, supplies the counter control signal to reset said counter to zero and configures said instruction supplying means to supply a first one of said sequential instructions.

22. The microprocessor of claim 21 further including means for determining whether a predefined condition exists within said microprocessor system, and

means for controlling response of said instruction decoding means to said SKIP instruction and said predefined condition to execute or not execute said SKIP instruction based on existence of said predefined condition

23. The microprocessor of claim 20 further comprising a loop counter, said instruction decoding means, responsive to a MICROLOOP instruction within said instruction register, providing a decrement signal to said loop counter and priding the counter control signal to reset said counter to zero, and said instruction supplying means being configured to supply from said instruction register said sequential instructions beginning with the first instruction in said instruction register, from a current one of said instruction groups, to said central processing unit.

24. The microprocessor of claim 23 further comprising:

means for determining whether a predefined condition exists within said microprocessor system, and

means for controlling response of said instruction decoding means to said MICROLOOP instruction and said predefined condition to execute or not execute said MICROLOOP instruction based on existence of said predefined condition.

25. The microprocessor of claim 20 wherein said instruction decoding means includes means, responsive to ones of said sequential instructions of predetermined type, for supplying control signals to said instruction fetching means such that a subsequent one of said instruction groups is provided to said instruction register.

26. The microprocessor of claim 25 wherein said instruction decoding means includes means for configuring said instruction supplying means to supply a remainder of a current one of said instruction groups within said instruction register as said operand to said central processing unit.

27. The microprocessor of claim 25 further comprising means for determining whether a predefined condition exists within said microprocessor system, and means for controlling response of said instruction decoding means to branch-type ones of said instructions and said predefined condition to execute or not execute said branch-type ones of said instructions based on existence of said predefined condition

28. The microprocessor of claim 20 wherein said instruction decoding means are configured to supply control signals to said instruction fetching means such that a subsequent one of said instruction groups is supplied as an operand in response to one of said sequential instructions.

29. In a microprocessor system including a central processing unit, memory, and an instruction register, a method for providing instructions and operands from said memory to said central processing unit comprising the steps of:

providing instruction groups to said instruction register from said memory wherein certain of said instruction groups include at least one instruction that, when executed, causes an access to an operand or an instruction or both said operand or instruction being located at a predetermined position from a boundary of said instruction groups;

decoding said at least one instruction to determine said predetermined position;

locating said predetermined position; and

supplying, from said instruction groups, using the predetermined location, said operand or instruction or both to said central processing unit.

* * * * *

CIVIL COVER SHEET

The JS 44 civil cover sheet and the information contained herein neither replace nor supplement the filing and service of pleadings or other papers as required by law, except as provided by local rules of court. This form, approved by the Judicial Conference of the United States in September 1974, is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet. (SEE INSTRUCTIONS ON THE REVERSE OF THE FORM.)

2-08 CV-174 JW

I. (a) PLAINTIFFS

Technology Properties Limited and Patriot Scientific Corporation

(b) County of Residence of First Listed Plaintiff _____
(EXCEPT IN U.S. PLAINTIFF CASES)

DEFENDANTS
HTC Corporation, HTC America, Inc. and ASUSTek Computer, Inc

County of Residence of First Listed Defendant _____
(IN U.S. PLAINTIFF CASES ONLY)
NOTE: IN LAND CONDEMNATION CASES, USE THE LOCATION OF THE LAND INVOLVED

(C) Attorney's (Firm Name, Address, and Telephone Number)

S. Calvin Capshaw
Capshaw DeRieux, LLP
1127 Judson Road, Suite 220
P.O. Box 3999 (75606-3999)
Longview, TX 75601
(903) 236-9800

Attorneys (If Known)

II. BASIS OF JURISDICTION (PLACE AN "X" IN ONE BOX ONLY)

- ☐ 1 U.S. Government Plaintiff
- ☒ 3 Federal Question (U.S. Government Not a Party)
- ☐ 2 U.S. Government Defendant
- ☐ 4 Diversity (Indicate Citizenship of Parties in Item III)

III. CITIZENSHIP OF PRINCIPAL PARTIES (Place an "X" in One Box for Plaintiff and One Box for Defendant)
(For Diversity Cases Only)

	PTF	DEF		PTF	DEF
Citizen of This State	☐ 1	☐ 1	Incorporated or Principal Place of Business in This State	☐ 4	☐ 4
Citizen of Another State	☐ 2	☐ 2	Incorporated and Principal Place of Business in Another State	☐ 5	☐ 5
Citizen or Subject of a Foreign Nation	☐ 3	☐ 3	Foreign Country	☐ 6	☐ 6

IV. NATURE OF SUIT (Place an "X" in One Box Only)

CONTRACT
- ☐ 110 Insurance
- ☐ 120 Marine
- ☐ 130 Miller Act
- ☐ 140 Negotiable Instrument
- ☐ 150 Recovery of Overpayment & Enforcement of Judgment
- ☐ 151 Medicare Act
- ☐ 152 Recovery of Defaulted Student Loans (Excl. Veterans)
- ☐ 153 Recovery of Overpayment of Veteran's Benefits
- ☐ 160 Stockholders' Suits
- ☐ 190 Other Contract
- ☐ 195 Contract Product Liability
- ☐ 196 Franchise

TORTS

PERSONAL INJURY
- ☐ 310 Airplane
- ☐ 315 Airplane Product Liability
- ☐ 320 Assault Libel & Slander
- ☐ 330 Federal Employers Liability
- ☐ 340 Marine
- ☐ 345 Marine Product Liability
- ☐ 350 Motor Vehicle
- ☐ 355 Motor Vehicle Product Liability
- ☐ 360 Other Personal Injury

PERSONAL INJURY
- ☐ 362 Personal Injury - Med. Malpractice
- ☐ 365 Personal Injury – Product Liability
- ☐ 368 Asbestos Personal Injury Product Liability

PERSONAL PROPERTY
- ☐ 370 Other Fraud
- ☐ 371 Truth in Lending
- ☐ 380 Other Personal Property Damage
- ☐ 385 Property Damage Product Liability

REAL PROPERTY
- ☐ 210 Land Condemnation
- ☐ 220 Foreclosure
- ☐ 230 Rent Lease & Ejectment
- ☐ 240 Torts to Land
- ☐ 245 Tort Product Liability
- ☐ 290 All Other Real Property

CIVIL RIGHTS
- ☐ 441 Voting
- ☐ 442 Employment
- ☐ 443 Housing Accommodations
- ☐ 444 Welfare
- ☐ 440 Other Civil Rights

PRISONER PETITIONS
- ☐ 510 Motions to Vacate Sentence

HABEAS CORPUS:
- ☐ 530 General
- ☐ 535 Death Penalty
- ☐ 540 Mandamus & Other
- ☐ 550 Civil Rights
- ☐ 555 Prison Condition

FORFEITURE/PENALTY
- ☐ 610 Agriculture
- ☐ 620 Other Food & Drug
- ☐ 625 Drug Related Seizure of Property 21 USC 881
- ☐ 630 Liquor Laws
- ☐ 640 R.R. & Truck
- ☐ 650 Airline Regs
- ☐ 660 Occupational Safety/Health
- ☐ 690 Other

LABOR
- ☐ 710 Fair Labor Standards Act
- ☐ 720 Labor/Mgmt. Relations
- ☐ 730 Labor/Mgmt. Reporting & Disclosure Act
- ☐ 740 Railway Labor Act
- ☐ 790 Other Labor Litigation
- ☐ 791 Empl. Ret. Inc Security Act

BANKRUPTCY
- ☐ 422 Appeal 28 USC 158
- ☐ 423 Withdrawal 28 USC 157

PROPERTY RIGHTS
- ☐ 820 Copyrights
- ☒ 830 Patent
- ☐ 840 Trademark

SOCIAL SECURITY
- ☐ 861 HIA (1 395ff)
- ☐ 862 Black Lung (923)
- ☐ 863 DIWC/DIWW (405(g))
- ☐ 864 SSID Title XVI
- ☐ 865 RSI (405(g))

FEDERAL TAX SUITS
- ☐ 870 Taxes (U.S. Plaintiff or Defendant
- ☐ 871 IRS – Third Party 26 USC 7609

OTHER STATUTES
- ☐ 400 State Reapportionment
- ☐ 410 Antitrust
- ☐ 430 Banks and Banking
- ☐ 450 Commerce
- ☐ 460 Deportation
- ☐ 470 Racketeer Influenced and Corrupt Organizations
- ☐ 480 Consumer Credit
- ☐ 490 Cable/Sat TV
- ☐ 810 Selective Service
- ☐ 850 Securities/Commodities/ Exchange
- ☐ 875 Customer Challenge 12 USC 3410
- ☐ 890 Other Statutory Actions
- ☐ 891 Agricultural Acts
- ☐ 892 Economic Stabilization Act
- ☐ 893 Environmental Matters
- ☐ 894 Energy Allocation Act
- ☐ 895 Freedom of Information Act
- ☐ 900 Appeal of Fee Determination Under Equal Access to Justice
- ☐ 950 Constitutionality of State State Statutes

V. ORIGIN (Place an "X" in One Box Only)

☒ 1 Original Proceeding | ☐ 2 Removed from State Court | ☐ 3 Remanded from Appellate Court | ☐ 4 Reinstated or Reopened | ☐ 5 Transferred from another district (specify) | ☐ 6 Multidistrict Litigation | ☐ 7 Appeal to District Judge from Magistrate Judgment

VI. CAUSE OF ACTION (Cite the U.S. Civil Statute Under which you are filing (Do not cite jurisdictional statutes unless diversity): **35 U.S.C. § 271**
(Brief description of cause:

VII. REQUESTED IN COMPLAINT: ☐ CHECK IF THIS IS A CLASS ACTION UNDER F.R.C.P. 23 | DEMAND $ | CHECK YES only if demanded in complaint: JURY DEMAND: ☒ YES ☐ NO

VII. RELATED CASE(S) (See Instructions): **IF ANY** See the attached page

DATE
April 25, 2008

SIGNATURE OF ATTORNEY OF RECORD

FOR OFFICE USE ONLY

Related Cases

1. Technology Properties Limited, Inc., et al v Fujitsu Limited, et al; In the United States District Court for the Eastern District of Texas, Marshall Division; Cause No 2:05-cv-00494 (TJW)

2. Technology Properties Limited, Inc., et al v. HTC Corporation, et al; In the United States District Court for the Eastern District of Texas, Marshall Division - Cause No. 2:08-cv-_____ - No cause number yet – filed simultaneously on April 25, 2008.

3 Technology Properties Limited, Inc , et al v HTC Corporation, et al; In the United States District Court for the Eastern District of Texas, Marshall Division - Cause No 2:08-cv-_____ - No cause number yet – filed simultaneously on April 25, 2008.

4. Technology Properties Limited, Inc., et al v. ASUSTek Computer, Inc ; In the United States District Court for the Eastern District of Texas, Marshall Division - Cause No 2:08-cv-_____ - No cause number yet – filed simultaneously on April 25, 2008

5. Technology Properties Limited, Inc., et al v. ASUSTek Computer, Inc.; In the United States District Court for the Eastern District of Texas, Marshall Division - Cause No. 2:08-cv-_____ - No cause number yet – filed simultaneously on April 25, 2008.

6 Technology Properties Limited, Inc , et al v. Acer, Inc., et al; In the United States District Court for the Eastern District of Texas, Marshall Division; Cause No 2:08-cv-_____ - No cause number yet – filed simultaneously on April 25, 2008.

7. Technology Properties Limited, Inc., et al v. Acer, Inc , et al; In the United States District Court for the Eastern District of Texas, Marshall Division; Cause No. 2:08-cv-_____ - No cause number yet – filed simultaneously on April 25, 2008.

EXHIBIT G

IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

(1) TECHNOLOGY PROPERTIES LIMITED and (2) PATRIOT SCIENTIFIC CORPORATION,

Plaintiffs,

vs.

(1) HTC CORPORATION and
(2) HTC AMERICA, INC ,

Defendants.

CASE NO 2-08CV-226

Jury Trial Demanded

COMPLAINT FOR PATENT INFRINGEMENTAND DEMAND FOR JURY TRIAL

Plaintiffs, Technology Properties Limited, Inc ("TPL") and Patriot Scientific Corporation ("Patriot"), (collectively "Plaintiffs"), allege the following in support of their Complaint for Patent Infringement and Demand for Jury Trial ("Complaint") against Defendants, HTC Corporation ("HTC") and HTC America, Inc. ("HTC America").

PARTIES

1. Plaintiff, Technology Properties Limited, Inc. ("TPL") is a corporation duly organized and existing under the laws of the State of California and maintains its principal place of business in San Jose, California.

2. Plaintiff, Patriot Scientific Corporation ("Patriot") is a corporation duly organized and existing under the laws of the State of Delaware and maintains its principal place of business in Carlsbad, California

3. Upon information and belief, Defendant HTC Corporation is a Taiwan corporation with its principal place of business in Taoyuan, Taiwan, R.O.C.

4. Upon information and belief, Defendant HTC America, Inc. is a Texas corporation with its principal place of business in Bellevue, Washington.

JURISDICTION

5. This Court has subject matter jurisdiction over this action pursuant to 28 U.S.C. §§ 1331, 1338(a) because this action arises under the patent laws of the United States, including 35 U.S.C. §§ 101, *et seq.* and 271, *et seq.* This Court has personal jurisdiction over Defendants because they each infringe Plaintiffs' patent by offering on their websites infringing products to their users and/or customers who reside in, or may be found in, the Eastern District of Texas. Further, each Defendant has actually transacted business with users of their websites in the Eastern District of Texas.

VENUE

6. Venue is proper in this judicial district under 28 U.S.C. §§ 1391(b) and (c) and 1400(b) because Defendants have each committed acts of infringement in this district.

GENERAL ALLEGATIONS

7. On June 25, 1996, United States Patent No. 5,530,890 ("'890 patent") entitled "High Performance, Low Cost Microprocessor" was duly and legally issued. All rights and interest in the '890 patent were assigned to Patriot Scientific Corporation. A true and correct copy of the '890 patent is attached hereto as Exhibit A.

8. TPL and Patriot are co-owners of the '890 patent. TPL has the exclusive right to enforce and license the '890 patent, and has standing to sue.

COUNT 1

(Patent infringement Against HTC Corporation)

9 Paragraphs 1-8 of the Complaint set forth above are incorporated herein by reference.

10. Upon information and belief Defendant HTC has infringed and continues to infringe under 35 U.S.C. § 271 the '890 patent.

11. HTC's acts of infringement have caused damage to Plaintiffs. Under 35 U.S.C. § 284, Plaintiffs are entitled to recover from HTC Corporation the damages sustained by Plaintiffs as a result of its infringement of the '890 patent. HTC's infringement of Plaintiffs' exclusive rights under the '890 patent will continue to damage Plaintiffs' business, causing irreparable harm, for which there is no adequate remedy of law, unless enjoined by this Court under 35 U.S.C. § 283.

12. Plaintiffs allege, on information and belief, that HTC's acts of infringement were willful and deliberate

COUNT 2

(Patent infringement Against HTC America, Inc.)

13 Paragraphs 1-8 of the Complaint set forth above are incorporated herein by reference.

14. Upon information and belief Defendant HTC America has infringed and continues to infringe under 35 U S C. § 271 the '890 patent.

15. HTC America's acts of infringement have caused damage to Plaintiffs. Under 35 U.S.C. § 284, Plaintiffs are entitled to recover from HTC America the damages sustained by Plaintiffs as a result of its infringement of the '890 patent. HTC America's infringement of

Plaintiffs' exclusive rights under the '890 patent will continue to damage Plaintiffs' business, causing irreparable harm, for which there is no adequate remedy of law, unless enjoined by this Court under 35 U.S.C. § 283.

16. Plaintiffs allege, on information and belief, that HTC America's acts of infringement were willful and deliberate.

PRAYER FOR RELIEF

WHEREFORE, Plaintiffs respectfully request that this Court enter judgment against Defendants as follows:

A. For judgment that Defendants, HTC Corporation and HTC America, Inc., have infringed and continue to infringe the '890 patent;

B. For permanent injunctions under 35 U.S.C. § 283 against Defendants and their directors, officers, employees, agents, subsidiaries, parents, attorneys, and all persons acting in concert, on behalf of, in joint venture, or in partnership with Defendants from further acts of infringement;

C. For damages to be paid by Defendants adequate to compensate Plaintiffs for their infringement, including interests, costs and disbursements as the Court may deem appropriate under 35 U.S.C. § 284;

D. For judgment finding that Defendants infringement was willful and deliberate, entitling Plaintiffs to increased damages under 35 U.S.C. § 284;

E. For judgment finding this to be an exceptional case against Defendants and awarding Plaintiffs attorney fees under 35 U.S.C. § 285; and,

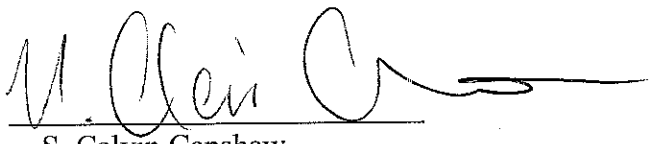
F. For such other and further relief at law and in equity as the court may deem just and proper.

DEMAND FOR JURY TRIAL

Pursuant to the Federal Rules of Civil Procedure Rule 38, Plaintiffs hereby demand a jury trial on all issues triable by jury.

Dated: June 4, 2008

Respectfully submitted,

By: 

S. Calvin Capshaw
State Bar No. 03783900
Email: ccapshaw@capshawlaw.com
Elizabeth L. DeRieux
State Bar No. 05770585
Email: ederieux@capshawlaw.com
N. Claire Abernathy
State Bar No. 24053063
E-mail: cherry@capshawlaw com
Capshaw DeRieux, LLP
1127 Judson Road, Suite 220
Longview, TX 75601
Telephone: (903) 236-9800
Facsimile: (903) 236-8787

Robert E. Krebs
California Bar No. 57526
Email: rkrebs@thelen.com
Christopher L. Ogden
California Bar No. 235517
Email: cogden@thelen.com
Thelen Reid Brown Raysman & Steiner, LLP
225 West Santa Clara Street, Suite 1200
San Jose, CA 95113-1723
Telephone: (408) 292-5800
Facsimile: (408) 287-8040

Ronald F. Lopez
California Bar No. 11756
Email: rflopez@thelen.com
Thelen Reid Brown Raysman & Steiner, LLP
101 Second Street, Suite 1800
San Francisco, CA 94105–3606
Telephone: (415) 371-1200
Facsimile: (415) 371-1211

ATTORNEYS FOR PLAINTIFF
TECHNOLOGY PROPERTIES LIMITED

By: R Chris Bunt by permission NCA

Robert M. Parker
State Bar No 15498000
Email: rmparker@pbatyler.com
Robert Christopher Bunt
State Bar No 00787165
Email: rcbunt@pbatyler.com
Parker, Bunt & Ainsworth, P.C.
100 East Ferguson, Ste 1114
Tyler, TX 75702
Telephone: (903) 531-3535
Facsimile: (903) 533-9687

Charles T. Hoge
California Bar No. 110696
Email: choge@knlh.com
Kirby Noonan Lance & Hoge, LLP
350 Tenth Avenue, Suite 1300
San Diego, CA 92101
Telephone: (619) 231-8666
Facsimile: (619) 231-9593

ATTORNEYS FOR PLAINTIFF
PATRIOT SCIENTIFIC CORPORATION

US005530890A

United States Patent [19]

Moore et al.

[11] Patent Number: 5,530,890

[45] Date of Patent: Jun. 25, 1996

[54] **HIGH PERFORMANCE, LOW COST MICROPROCESSOR**

[75] Inventors: **Charles H. Moore**, Woodside; **Russell H. Fish, III** Mt. View both of Calif.

[73] Assignee: **Nanotronics Corporation**, Eagle Point, Oreg.

[21] Appl. No.: **480,206**

[22] Filed: **Jun. 7, 1995**

Related U.S. Application Data

[62] Division of Ser. No. 389,334, Aug. 3, 1989, Pat. No. 5,440,749

[51] Int. Cl.[6] **G06F 9/22**

[52] U.S. Cl. **395/800**; 364/931; 364/925.6; 364/937.1; 364/965.4; 364/232.8; 364/244.3

[58] Field of Search 395/375, 500, 395/775, 800

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,603,934	9/1971	Heath	395/181
4,003,033	1/1977	O'Keefe et al.	395/287
4,037,090	7/1977	Raymond	364/706
4,042,972	8/1977	Grunes et al.	395/375
4,050,058	9/1977	Garlic	395/800
4,067,058	1/1978	Derchak	395/740
4,079,455	3/1978	Ozga	395/800
4,110,822	8/1978	Porter	395/375
4,125,871	11/1978	Martin	395/550
4,128,873	12/1978	Lamiaux	395/183.06
4,253,785	3/1981	Chamberlin	375/375
4,354,228	10/1982	Moore et al.	395/800
4,376,977	3/1983	Brunshorst	395/375
4,382,279	5/1983	Mgon	395/800
4,403,303	9/1983	Howes et al.	395/500
4,450,519	5/1984	Guttag et al.	395/800
4,463,421	7/1984	Laws	395/325
4,538,239	8/1985	Magar	364/759

(List continued on next page.)

OTHER PUBLICATIONS

C. Whitby–Strevans, "The transputer", *The 12th Annual International Symposium on Computer Architecture Conference Proceedings*, Jun. 17–19, 1985, pp. 292–300.

D. W. Best et al., "An Advanced–Architecture CMOS/SOS Microprocessor" *IEEE Micro*, vol. 2, No. 3, Aug. 1982, pp. 11–25.

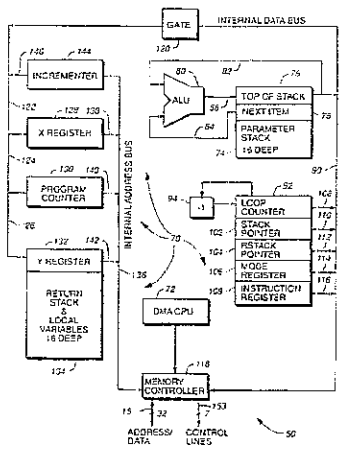
Primary Examiner—David Y. Eng

Attorney, Agent, or Firm—Cooley Godward Castro Huddleson & Tatum

[57] **ABSTRACT**

A microprocessor (**50**) includes a main central processing unit (CPU) (**70**) and a separate direct memory access (DMA) CPU (**72**) in a single integrated circuit making up the microprocessor (**50**). The main CPU (**70**) has a first 16 deep push down stack (**74**), which has a top item register (**76**) and a next item register (**78**), respectively connected to provide inputs to an arithmetic logic unit (ALU) (**80**) by lines (**82**) and (**84**). An output of the ALU (**80**) is connected to the top item register (**76**) by line (**86**). The output of the top item register at (**82**) is also connected by line (**88**) to an internal data bus (**90**). A loop counter (**92**) is connected to a decrementer (**94**) by lines (**96**) and (**98**). The loop counter (**92**) is bidirectionally connected to the internal data bus (**90**) by line (**100**). Stack pointer (**102**), return stack pointer (**104**), mode register (**106**) and instruction register (**108**) are also connected to the internal data bus (**90**) by lines (**110**), (**112**), (**114**) and (**116**), respectively. The internal data bus (**90**) is connected to memory controller (**118**) and to gate (**120**). The gate (**120**) provides inputs on lines (**122**), (**124**), and (**126**) to X register (**128**), program counter (**130**) and Y register (**132**) of return push down stack (**134**). The X register (**128**), program counter (**130**) and Y register (**132**) provide outputs to internal address bus (**136**) on lines (**138**), (**140**) and (**142**). The internal address bus provides inputs to the memory controller (**118**) and to an incrementer (**144**). The incrementer (**144**) provides inputs to the X register, program counter and Y register via lines (**146**), (**122**), (**124**) and (**126**). The DMA CPU (**72**) provides inputs to the memory controller (**118**) on line (**148**). The memory controller (**118**) is connected to a RAM by address/data bus (**150**) and control lines (**152**).

10 Claims, 19 Drawing Sheets



U.S. PATENT DOCUMENTS

4,541,045	9/1985	Kromer	395/375
4,562,537	12/1985	Barnett et al	395/375
4,577,282	3/1986	Candel et al	395/800
4,607,332	8/1986	Goldberg	395/375
4,626,988	12/1986	George et al	395/375
4,649,471	3/1987	Briggs	395/325
4,665,495	5/1987	Thaden	345/185
4,709,329	11/1987	Hecker	395/275
4,713,749	12/1987	Magar et al	395/375
4,714,994	12/1987	Oklobdzija et al	395/375
4,720,812	1/1988	Kao et al	395/700
4,772,888	9/1988	Kimura	340/825.5
4,777,591	10/1988	Chang et al	395/800
4,787,032	11/1988	Culley et al	395/725
4,803,621	2/1989	Kelly	395/400
4,860,198	8/1989	Takenaka	395/307
4,870,562	9/1989	Kimoto	395/550
4,931,986	6/1990	Daniel et al	395/550
5,036,460	7/1991	Takahira	395/425
5,070,451	12/1991	Moore et al	395/375
5,127,091	6/1992	Bonfarah	395/375

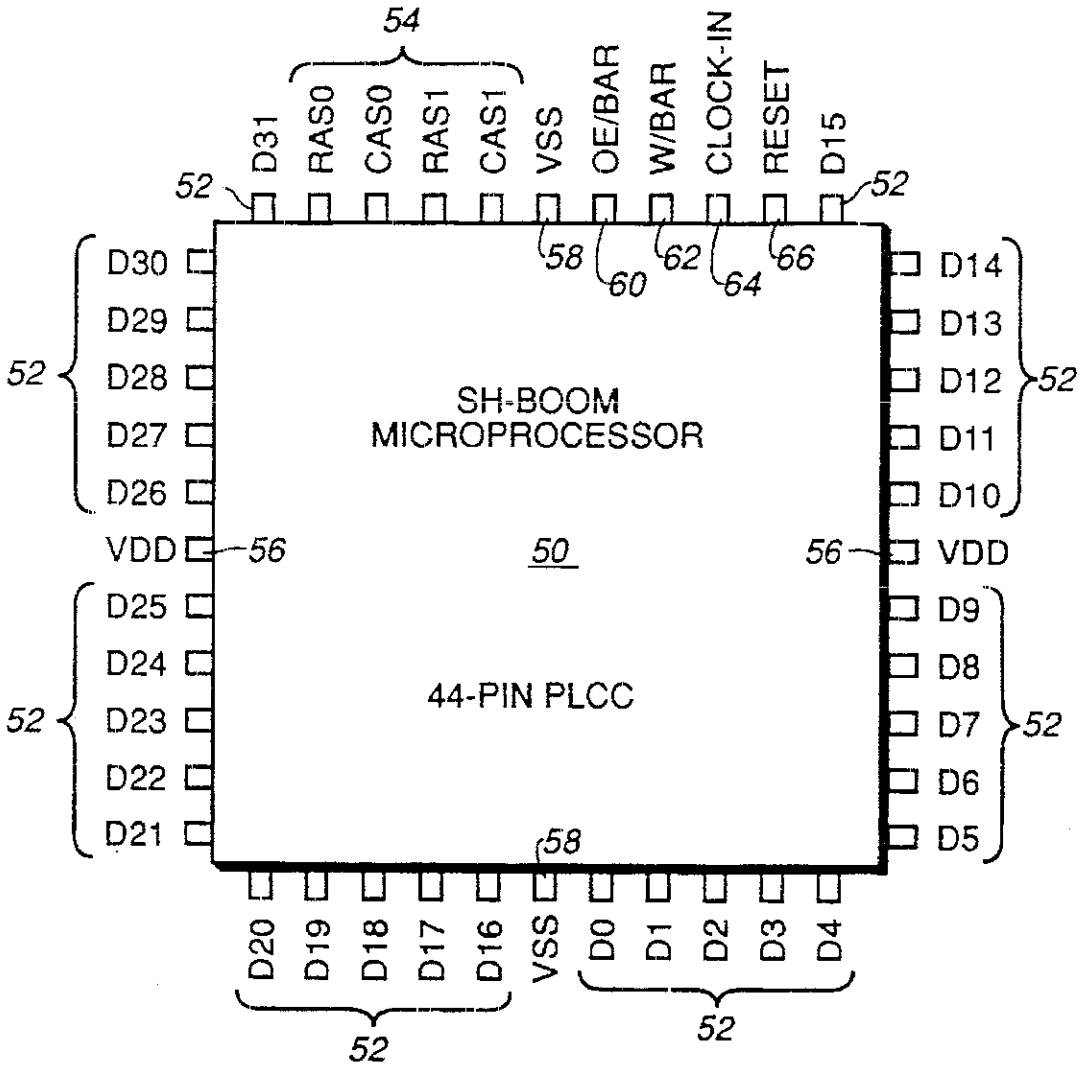

54
D31
RAS0
CAS0
RAS1
CAS1
VSS
OE/BAR
W/BAR
CLOCK-IN
RESET
D15
52
58
60
62
64
66
D30
D29
D28
D27
D26
VDD
D25
D24
D23
D22
D21
SH-BOOM
MICROPROCESSOR
50
44-PIN PLCC
56
D14
D13
D12
D11
D10
D9
D8
D7
D6
D5
D20
D19
D18
D17
D16
D0
D1
D2
D3
D4

FIG._1

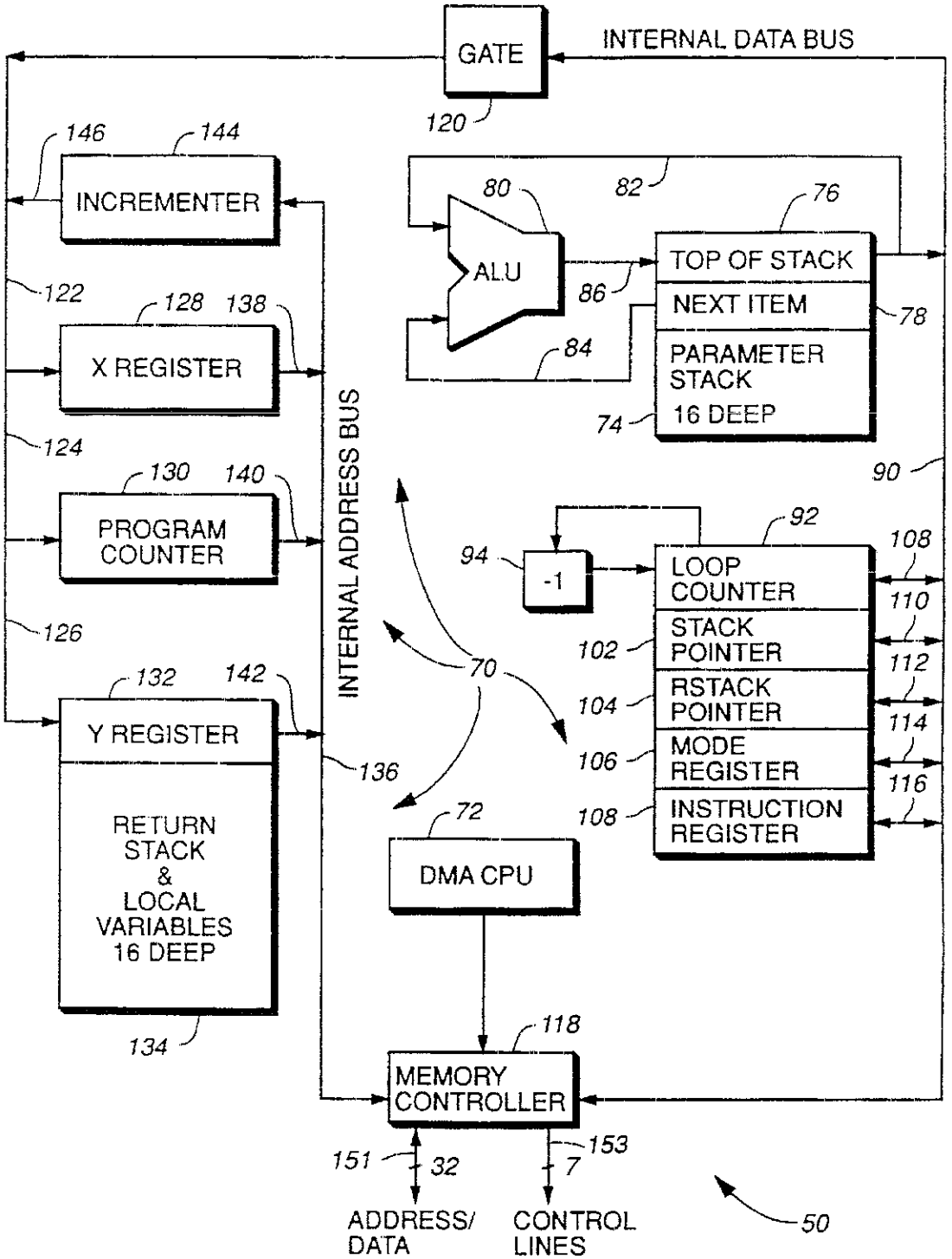

GATE
INTERNAL DATA BUS
120
146
144
INCREMENTER
80
82
76
ALU
TOP OF STACK
NEXT ITEM
PARAMETER STACK
16 DEEP
86
84
78
74
122
128
138
X REGISTER
124
130
140
PROGRAM COUNTER
126
132
142
Y REGISTER
RETURN STACK & LOCAL VARIABLES 16 DEEP
134
136
INTERNAL ADDRESS BUS
70
90
92
94
-1
LOOP COUNTER
STACK POINTER
RSTACK POINTER
MODE REGISTER
INSTRUCTION REGISTER
102
104
106
108
110
112
114
116
72
DMA CPU
118
MEMORY CONTROLLER
151
32
153
7
ADDRESS/ DATA
CONTROL LINES
50

FIG._2

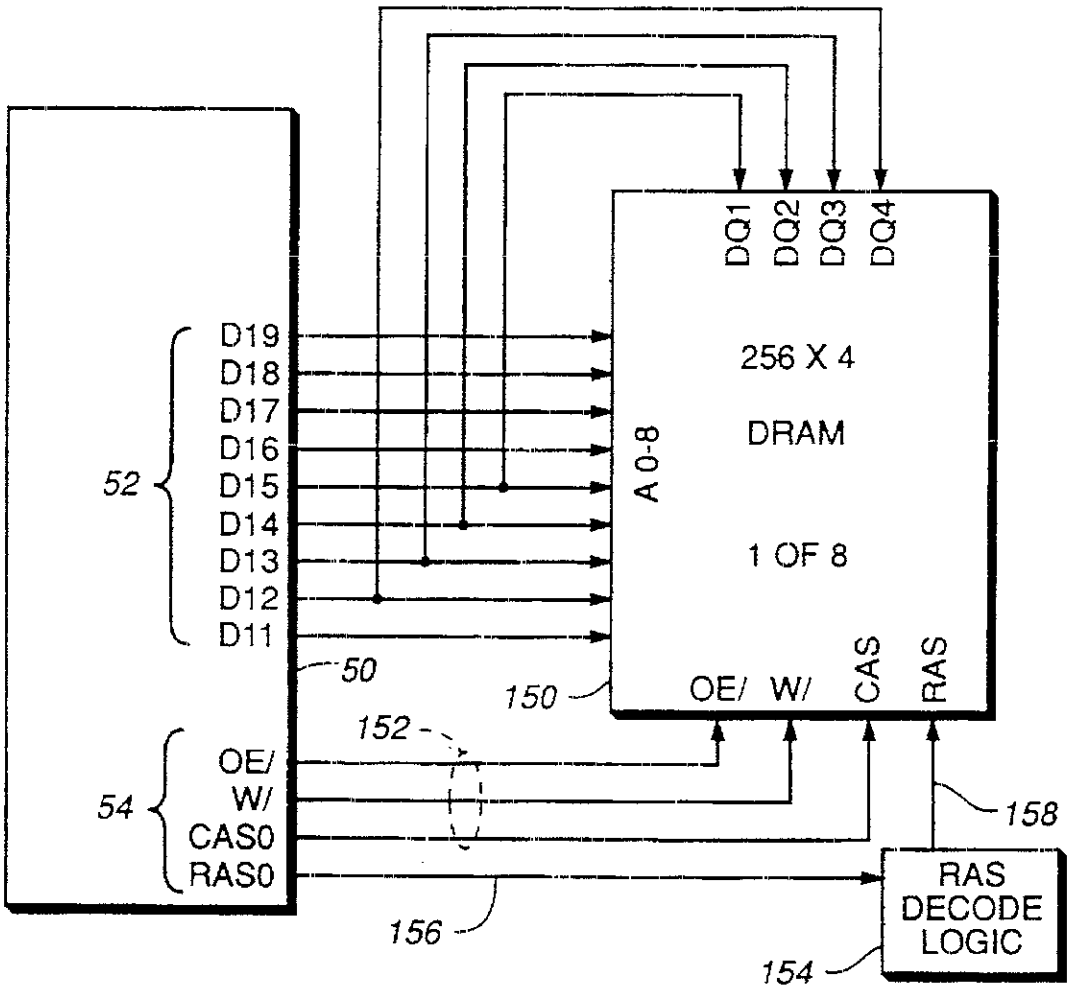

DQ1
DQ2
DQ3
DQ4
256 X 4
DRAM
1 OF 8
A 0-8
OE/
W/
CAS
RAS
D19
D18
D17
D16
D15
D14
D13
D12
D11
52
50
150
152
OE/
W/
CAS0
RAS0
54
158
RAS
DECODE
LOGIC
156
154

FIG._3

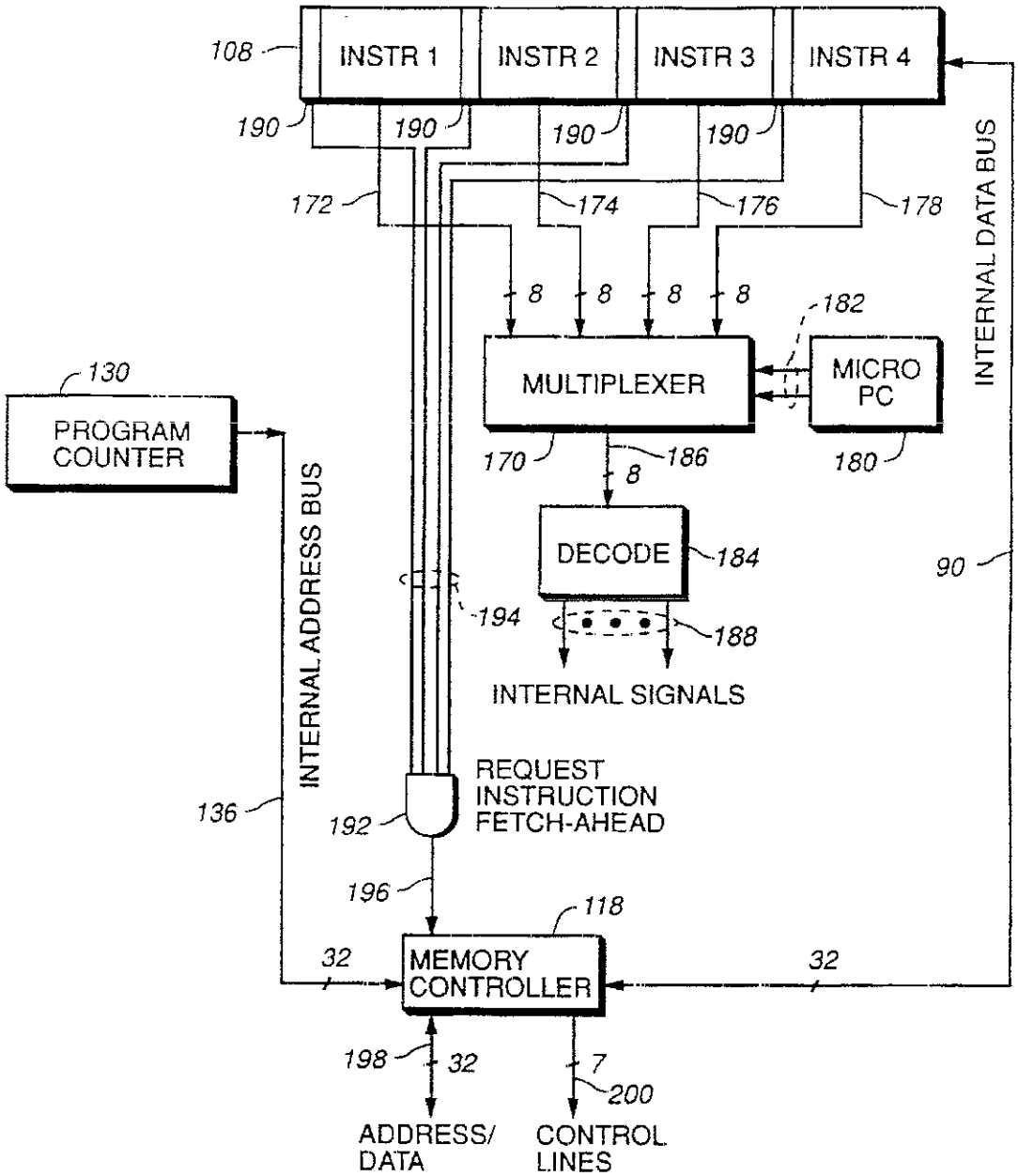

108
INSTR 1
INSTR 2
INSTR 3
INSTR 4
190
190
190
190
172
174
176
178
8
8
8
8
182
MULTIPLEXER
MICRO PC
INTERNAL DATA BUS
130
PROGRAM COUNTER
170
186
180
8
DECODE
184
90
194
188
INTERNAL SIGNALS
INTERNAL ADDRESS BUS
REQUEST INSTRUCTION FETCH-AHEAD
136
192
196
118
32
MEMORY CONTROLLER
32
198
32
7
200
ADDRESS/ DATA
CONTROL LINES

FIG._4

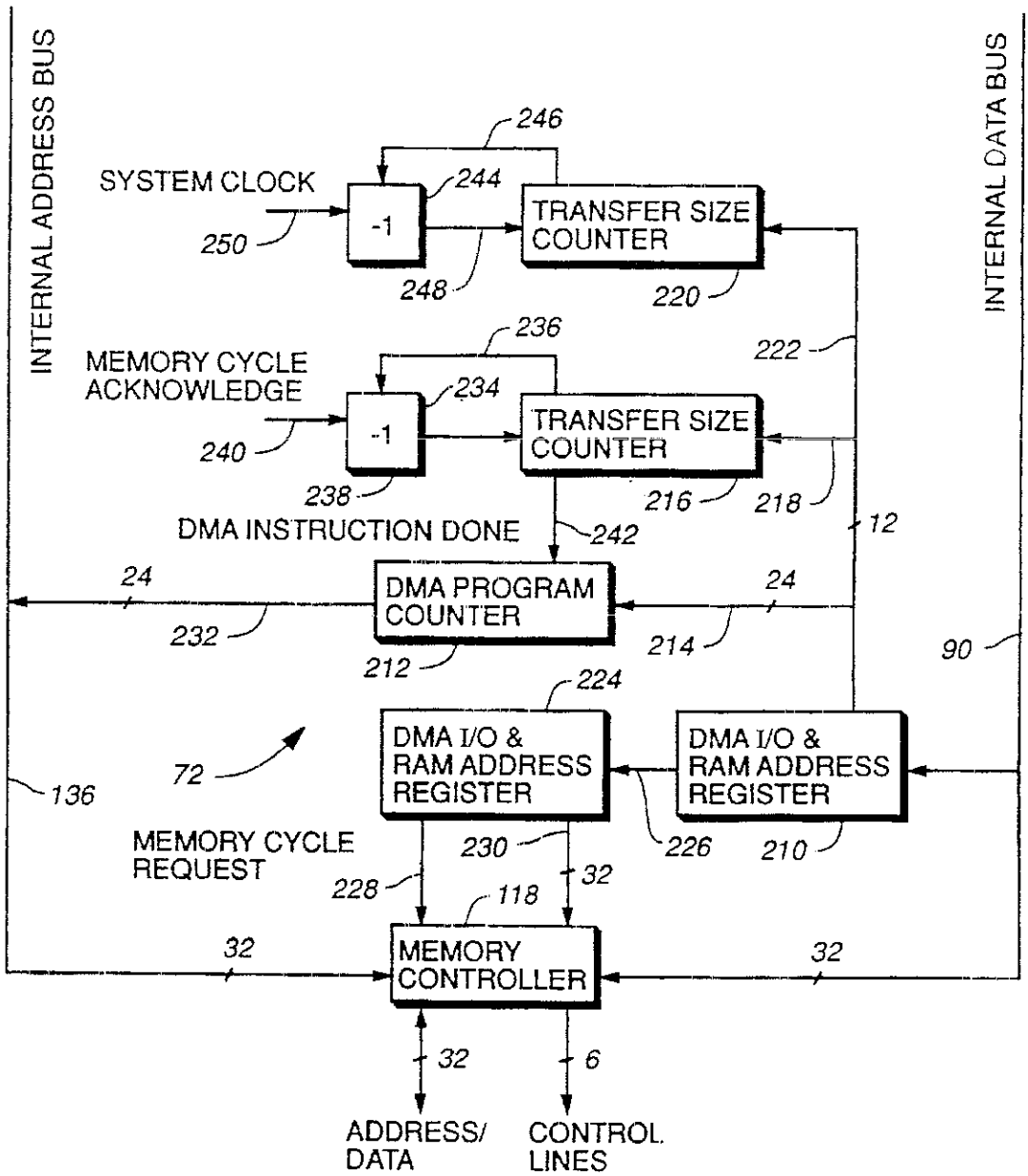

INTERNAL ADDRESS BUS
INTERNAL DATA BUS
SYSTEM CLOCK
250
246
244
-1
248
TRANSFER SIZE COUNTER
220
MEMORY CYCLE ACKNOWLEDGE
240
236
234
-1
238
TRANSFER SIZE COUNTER
216
218
222
12
DMA INSTRUCTION DONE
242
24
232
DMA PROGRAM COUNTER
212
24
214
90
224
DMA I/O & RAM ADDRESS REGISTER
DMA I/O & RAM ADDRESS REGISTER
72
136
MEMORY CYCLE REQUEST
230
226
210
228
32
118
32
MEMORY CONTROLLER
32
32
6
ADDRESS/ DATA
CONTROL LINES

FIG._5

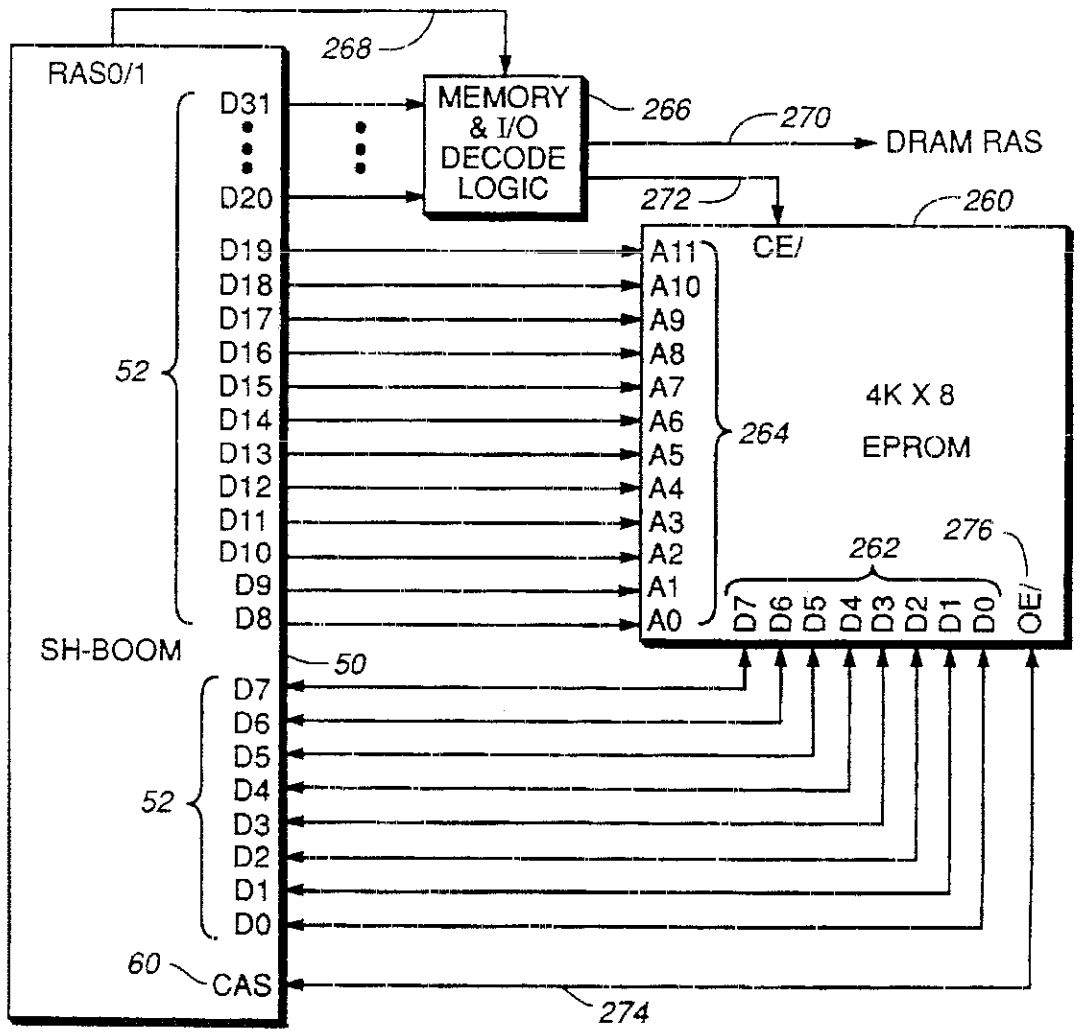

RAS0/1
268
MEMORY & I/O DECODE LOGIC
266
270
DRAM RAS
272
260
CE/
D31
D20
D19
D18
D17
D16
D15
D14
D13
D12
D11
D10
D9
D8
52
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
264
4K X 8
EPROM
262
276
D7
D6
D5
D4
D3
D2
D1
D0
OE/
SH-BOOM
50
52
60
CAS
274

FIG._6

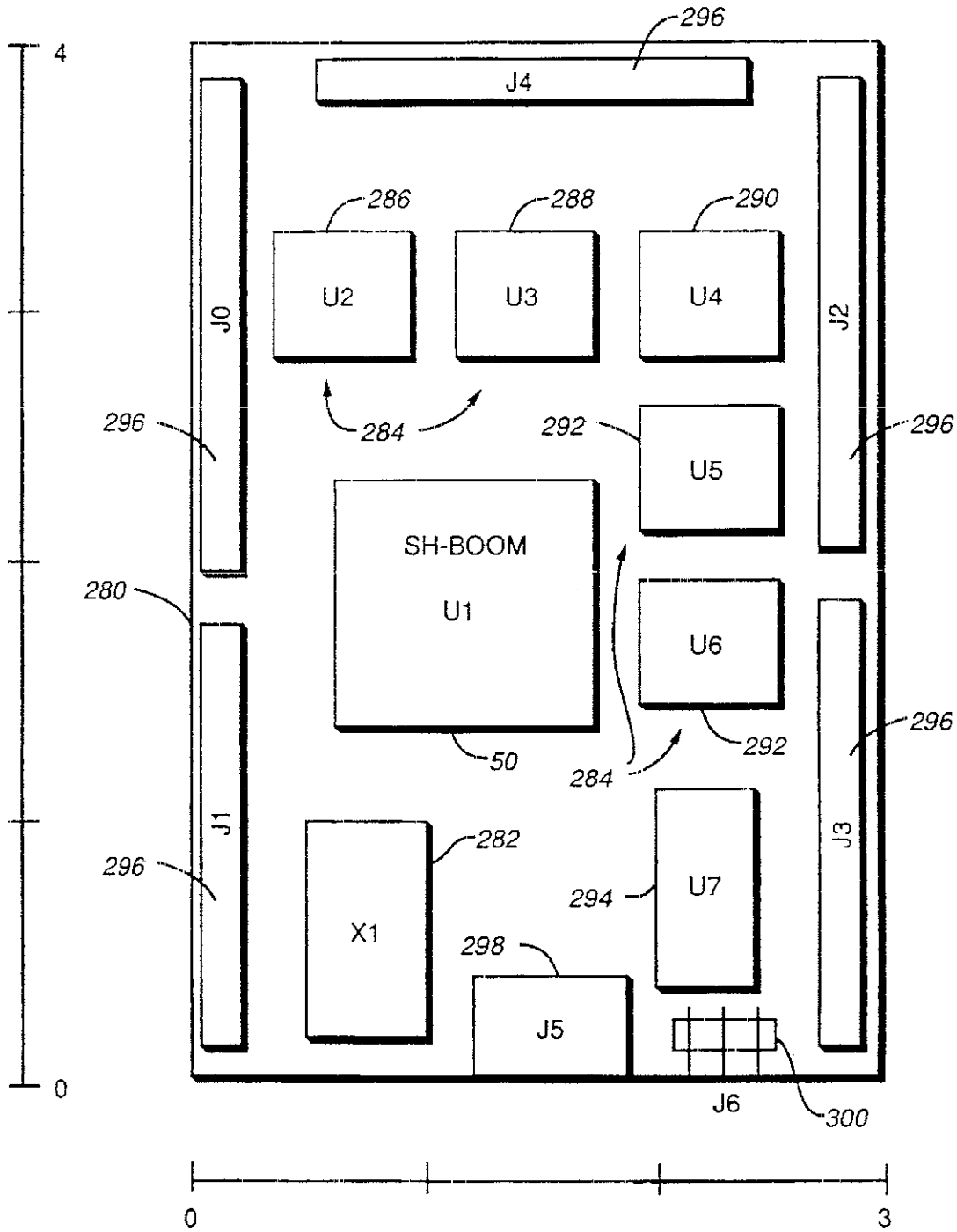

296
4
J4
286
288
290
U2
U3
U4
J0
J2
296
284
292
296
U5
SH-BOOM
U1
280
U6
296
50
284
292
J1
J3
282
296
294
U7
X1
298
J5
J6
300
0
0
3

FIG._7

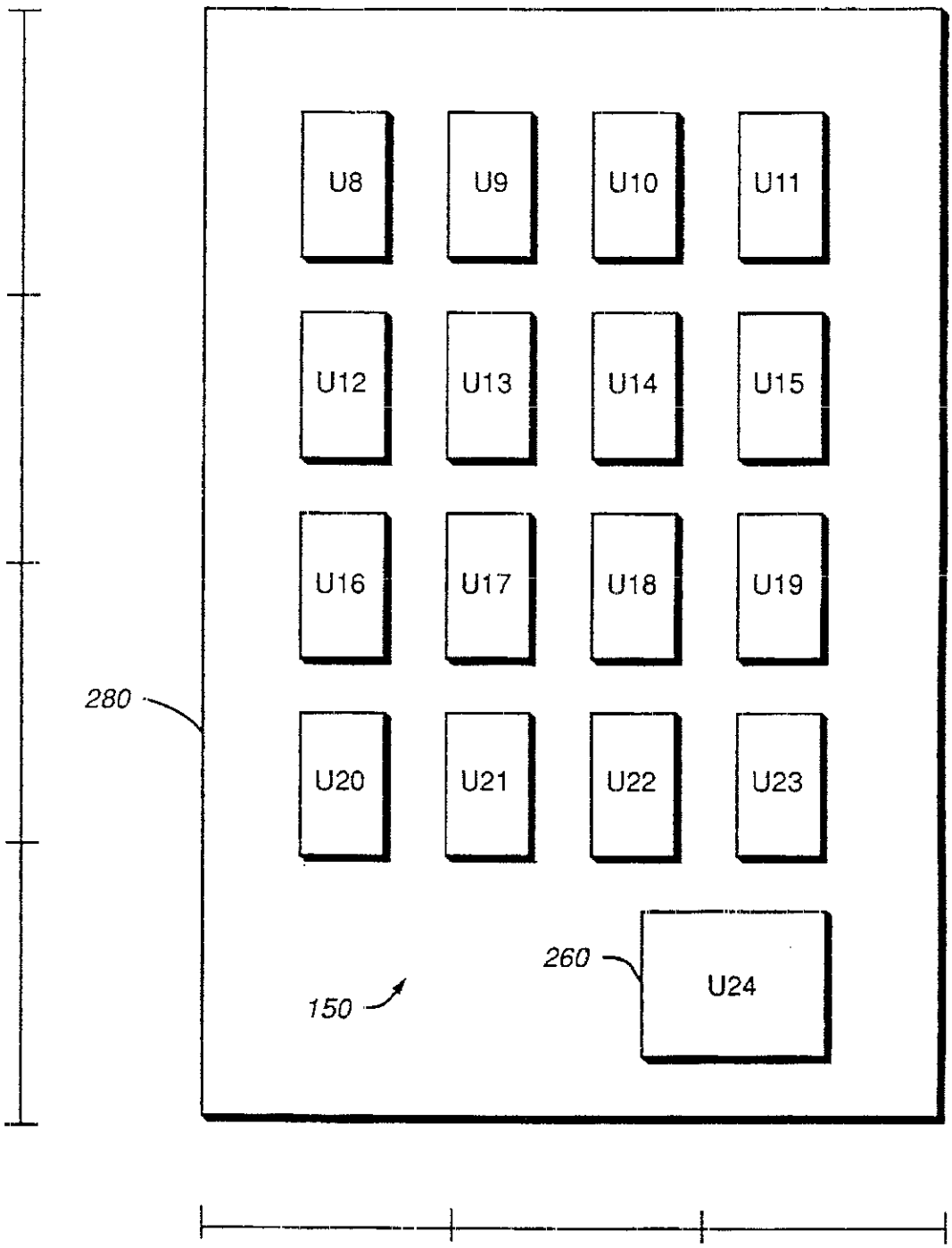

U8
U9
U10
U11
U12
U13
U14
U15
U16
U17
U18
U19
280
U20
U21
U22
U23
260
U24
150

FIG._8

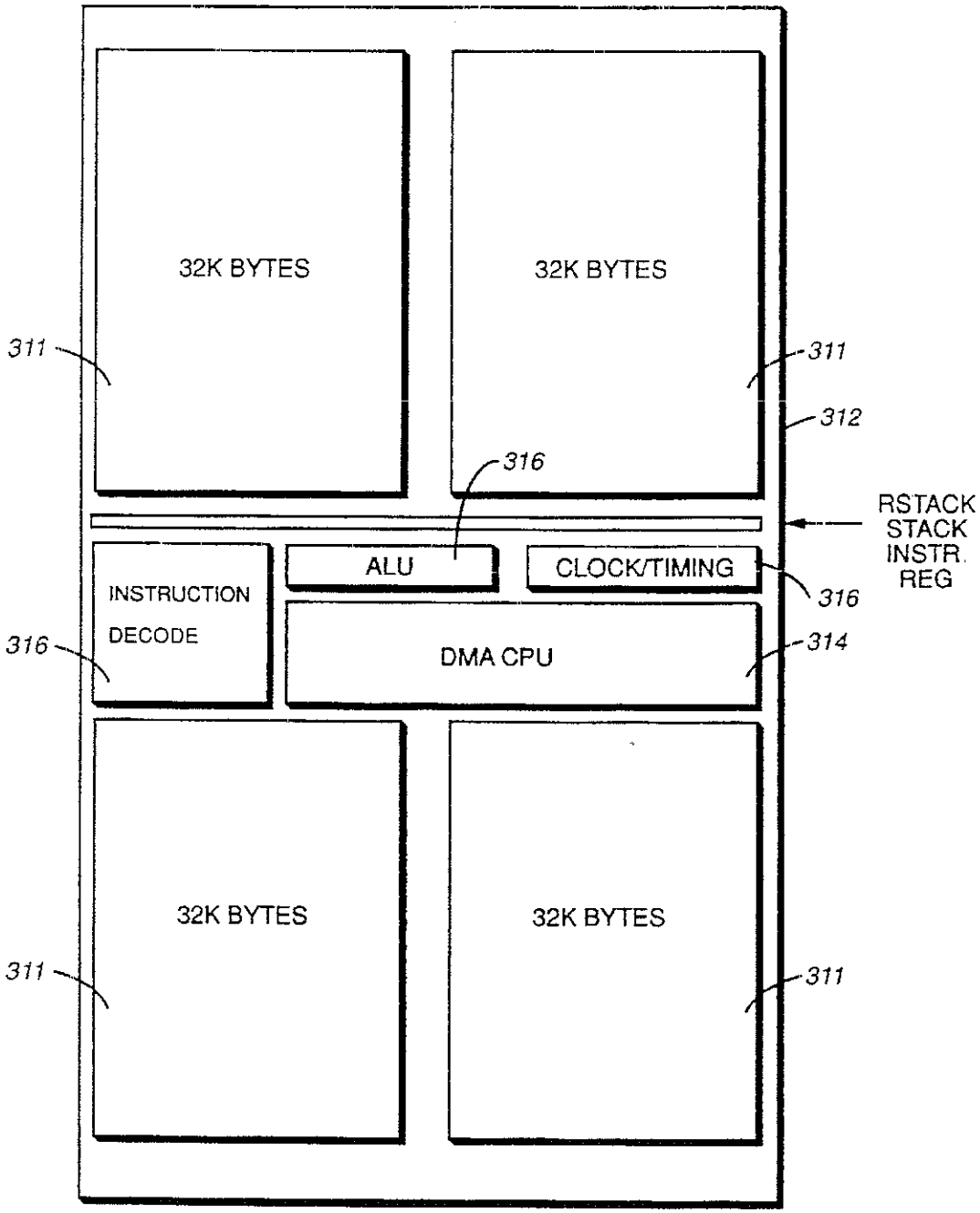

32K BYTES
32K BYTES
311
311
312
316
RSTACK
STACK
INSTR.
REG
ALU
CLOCK/TIMING
316
INSTRUCTION
DECODE
316
DMA CPU
314
32K BYTES
32K BYTES
311
311

FIG._9

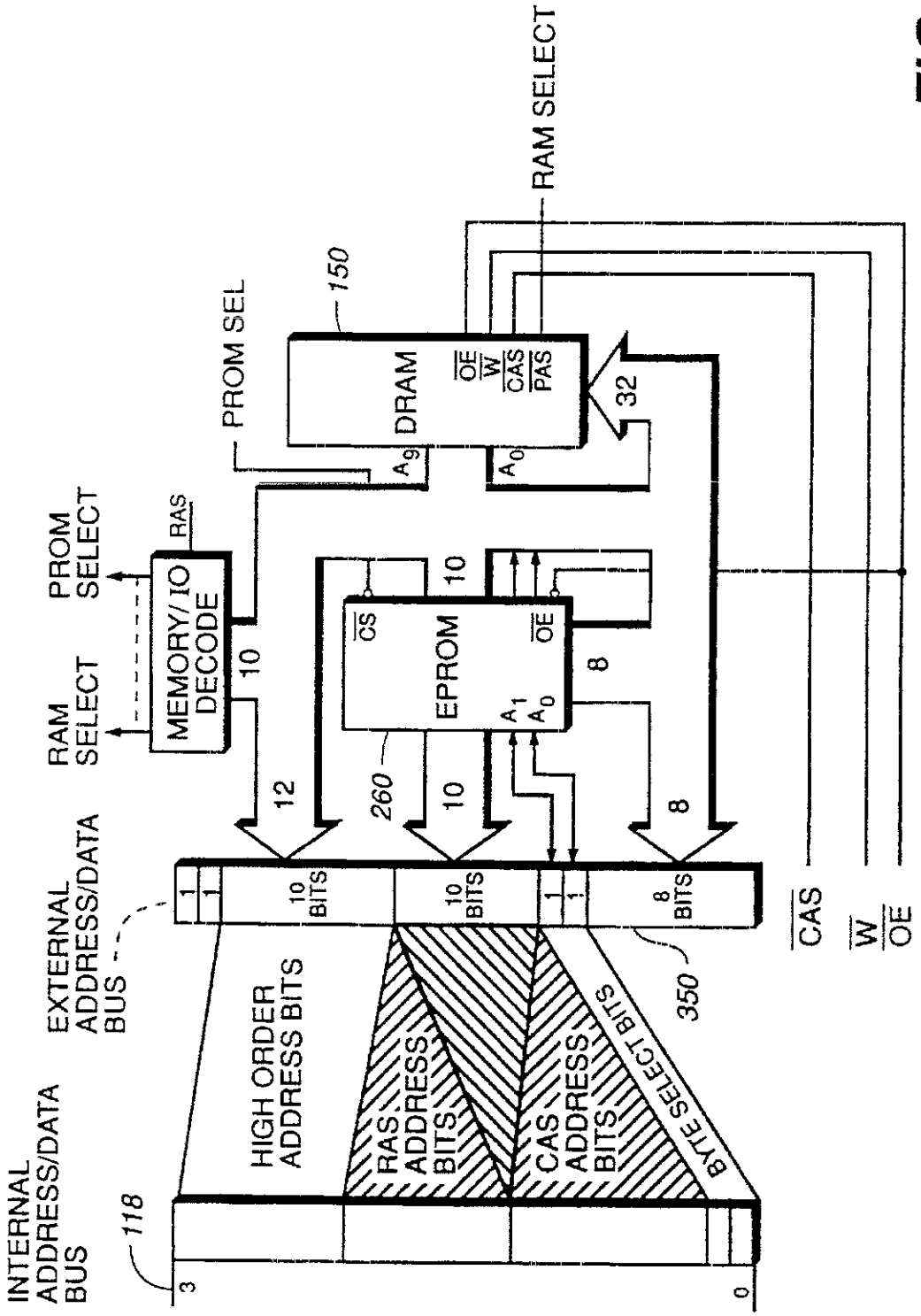
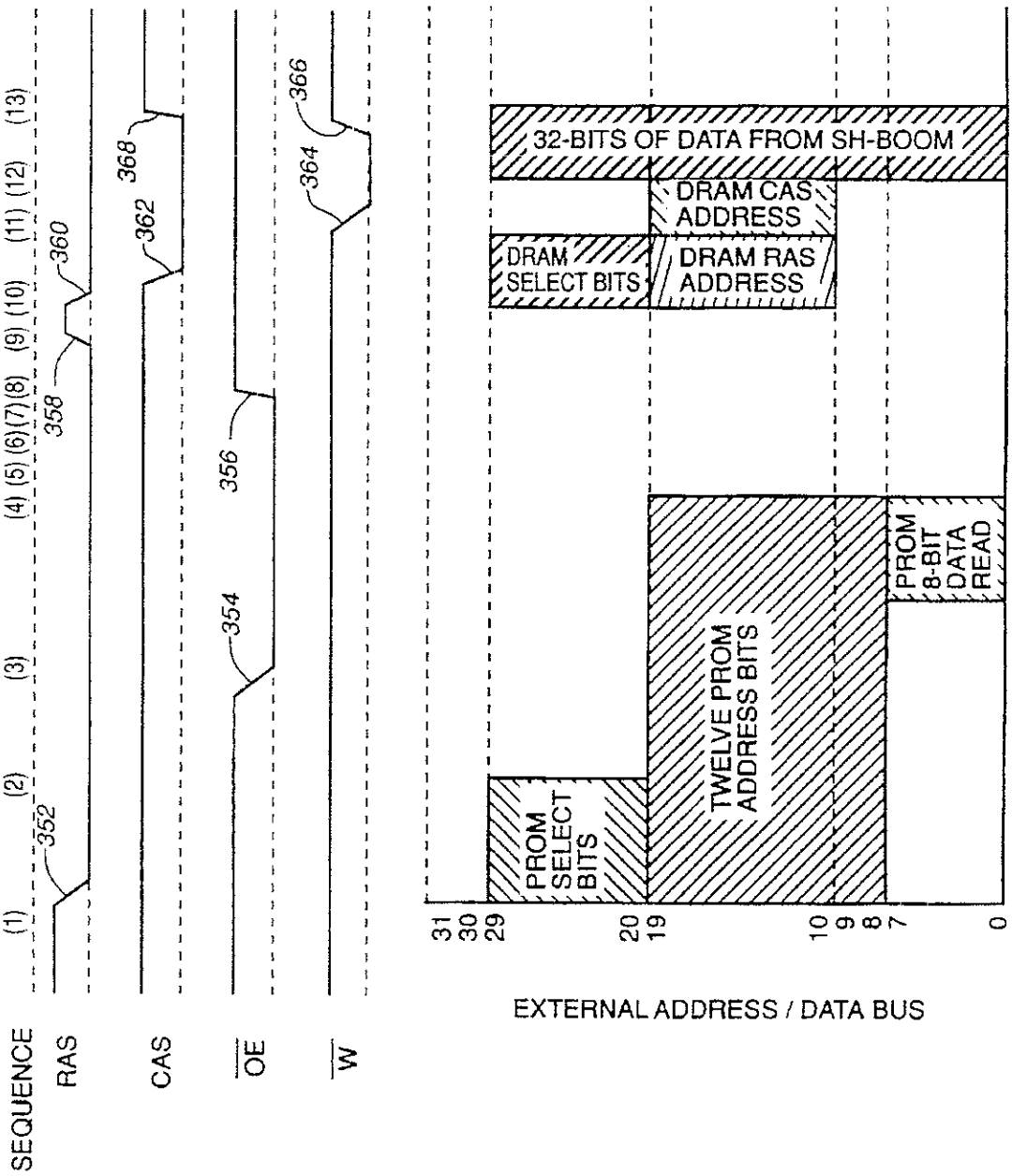

RAM SELECT
PROM SEL
150
DRAM
OE
W
CAS
PAS
32
A9
A0
PROM SELECT
RAS
MEMORY/IO DECODE
10
RAM SELECT
CS
EPROM
OE
A1
A0
10
8
12
260
10
8
EXTERNAL ADDRESS/DATA BUS
1
1
10 BITS
10 BITS
1
1
8 BITS
CAS
W
OE
350
HIGH ORDER ADDRESS BITS
RAS ADDRESS BITS
CAS ADDRESS BITS
BYTE SELECT BITS
INTERNAL ADDRESS/DATA BUS
118
3
0

FIG._10

FIG._11


SEQUENCE
RAS
CAS
OE
W
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
(12)
(13)
352
354
356
358
360
362
364
366
368
32-BITS OF DATA FROM SH-BOOM
DRAM CAS ADDRESS
DRAM SELECT BITS
DRAM RAS ADDRESS
PROM SELECT BITS
TWELVE PROM ADDRESS BITS
PROM 8-BIT DATA READ
31
30
29
20
19
10
9
8
7
0
EXTERNAL ADDRESS / DATA BUS

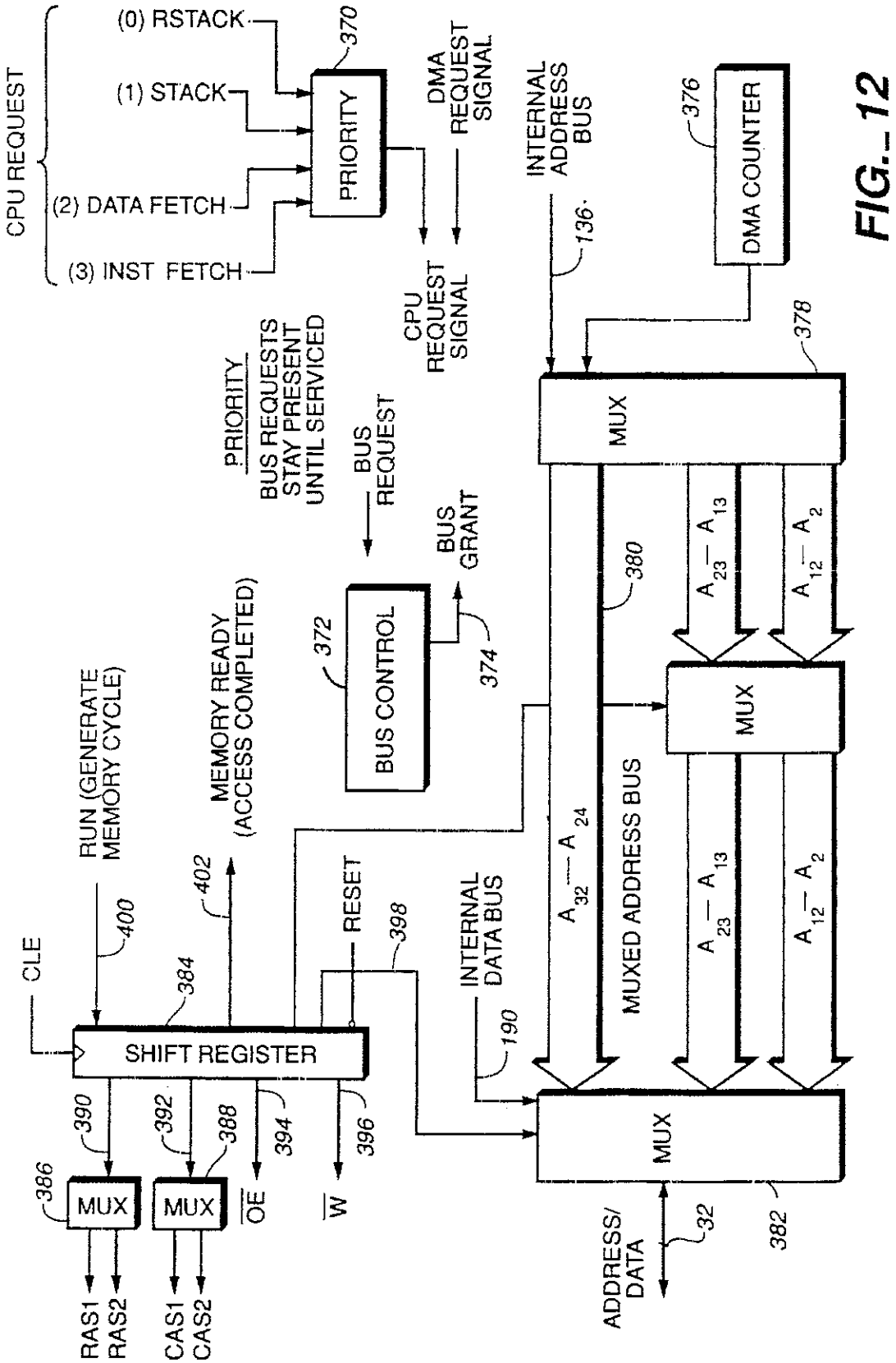

CPU REQUEST
(0) RSTACK
(1) STACK
(2) DATA FETCH
(3) INST FETCH
370
PRIORITY
DMA REQUEST SIGNAL
CPU REQUEST SIGNAL
INTERNAL ADDRESS BUS
136
376
DMA COUNTER
378
MUX
PRIORITY
BUS REQUESTS STAY PRESENT UNTIL SERVICED
BUS REQUEST
BUS GRANT
372
BUS CONTROL
374
380
A23 — A13
A12 — A2
MUX
MEMORY READY (ACCESS COMPLETED)
RUN (GENERATE MEMORY CYCLE)
402
400
RESET
398
INTERNAL DATA BUS
A32 — A24
MUXED ADDRESS BUS
CLE
384
SHIFT REGISTER
190
MUX
390
392
388
394
396
386
MUX
MUX
OE
W
ADDRESS/ DATA
32
382
RAS1
RAS2
CAS1
CAS2

FIG._12

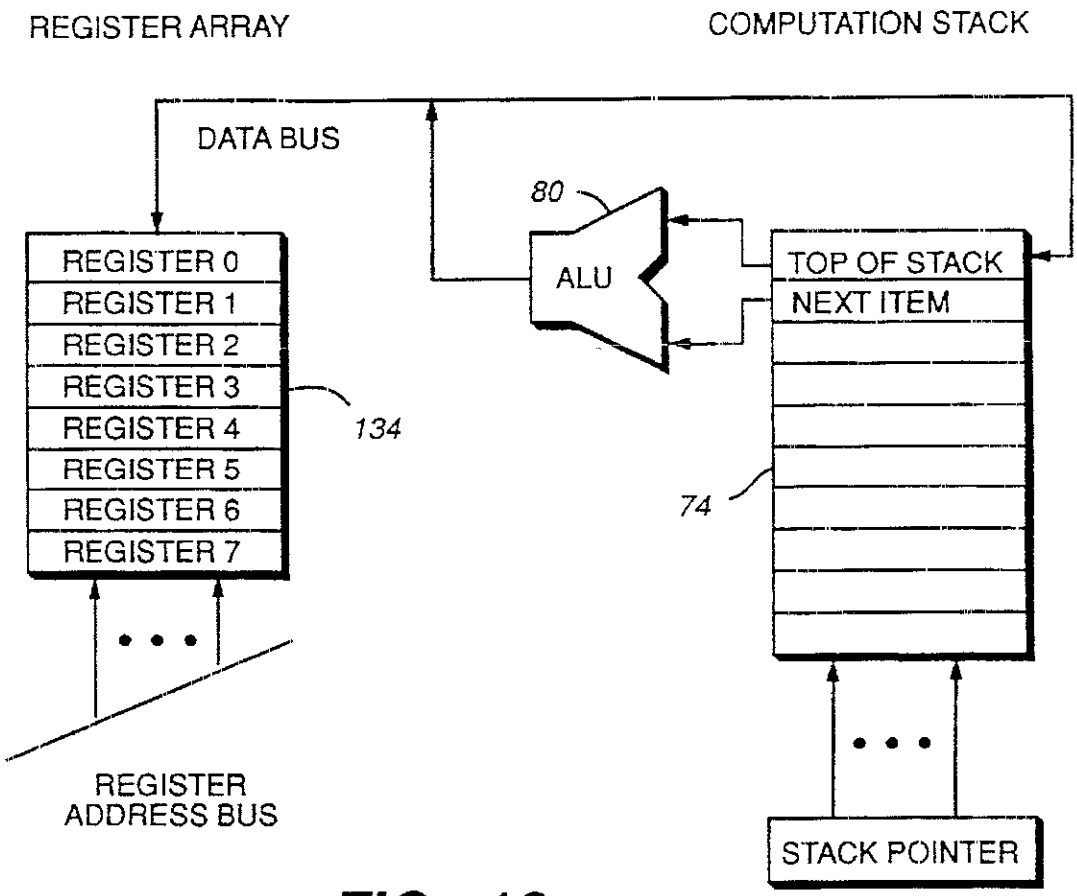

REGISTER ARRAY
COMPUTATION STACK
DATA BUS
80
ALU
REGISTER 0
REGISTER 1
REGISTER 2
REGISTER 3
REGISTER 4
REGISTER 5
REGISTER 6
REGISTER 7
134
TOP OF STACK
NEXT ITEM
74
REGISTER
ADDRESS BUS
STACK POINTER

FIG._13

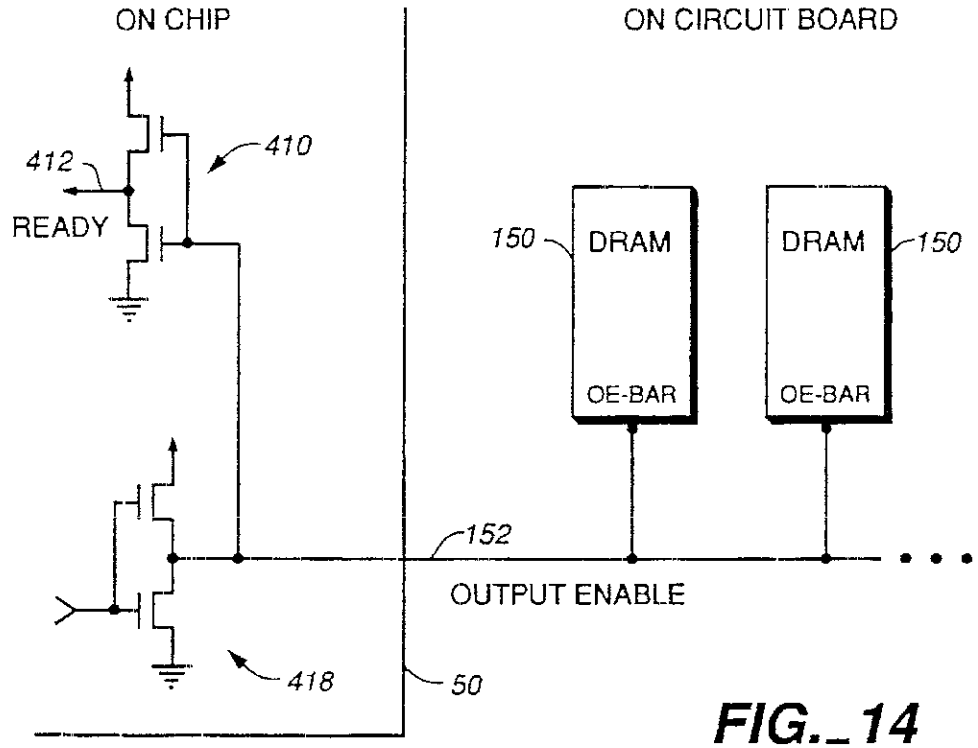

ON CHIP
ON CIRCUIT BOARD
412
410
READY
150
DRAM
OE-BAR
DRAM
OE-BAR
150
152
OUTPUT ENABLE
418
50

FIG._14

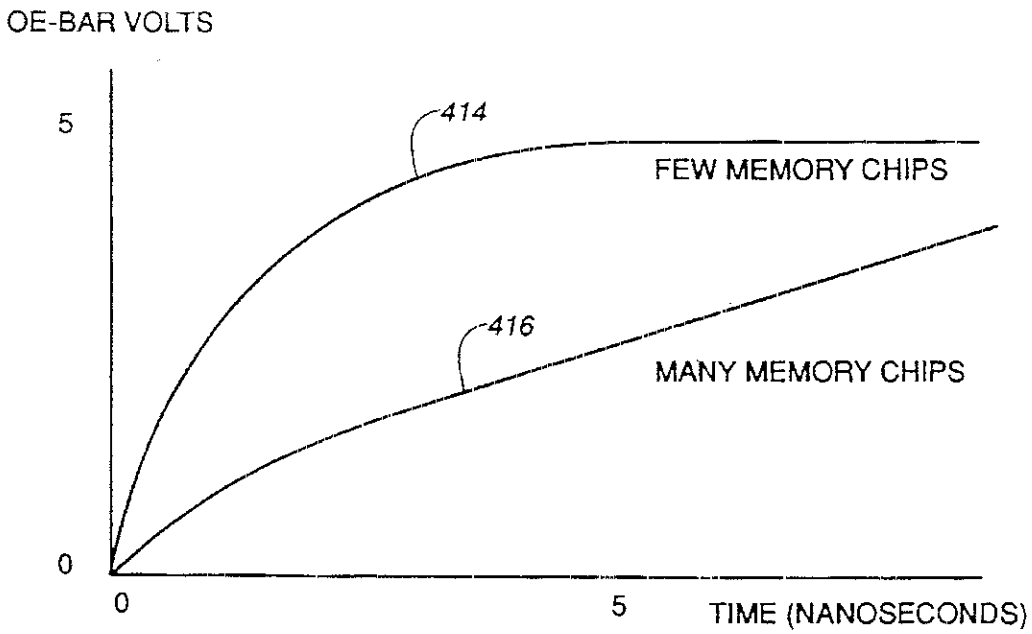

OE-BAR VOLTS
5
0
414
FEW MEMORY CHIPS
416
MANY MEMORY CHIPS
0
5
TIME (NANOSECONDS)

FIG._15

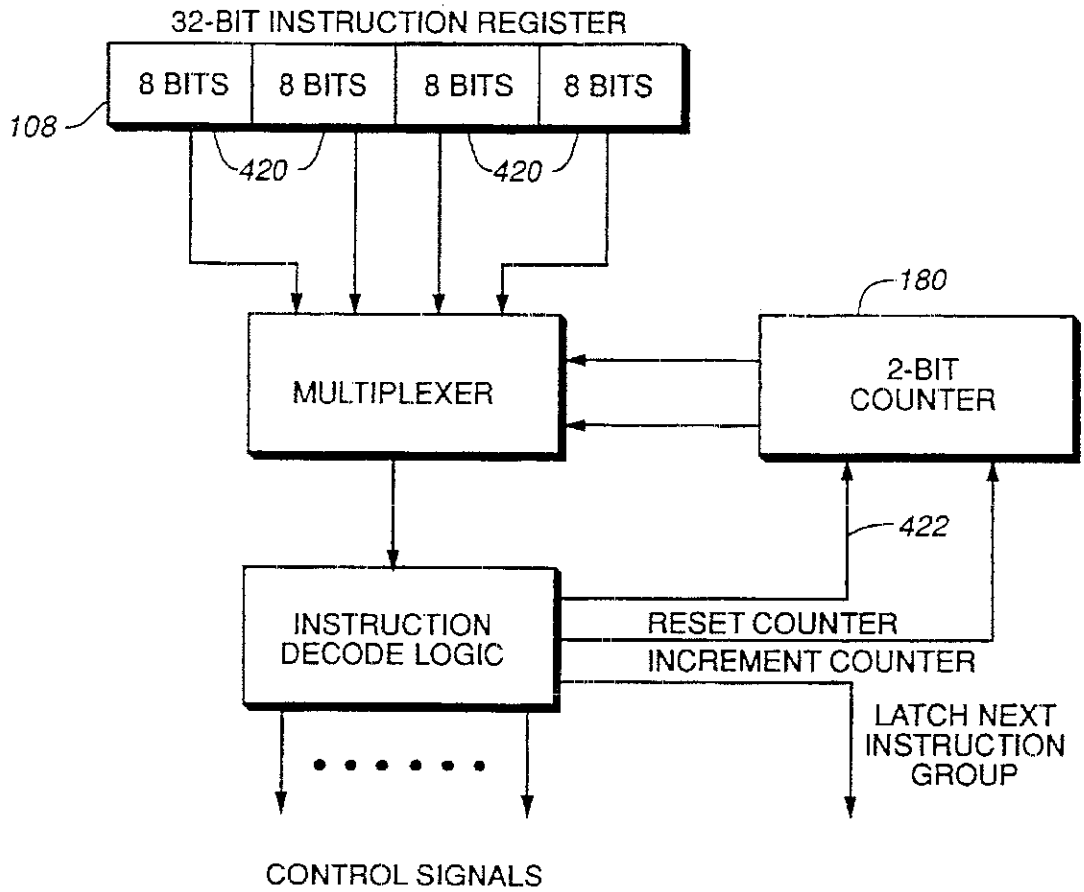

32-BIT INSTRUCTION REGISTER
8 BITS
8 BITS
8 BITS
8 BITS
108
420
420
180
MULTIPLEXER
2-BIT COUNTER
422
INSTRUCTION DECODE LOGIC
RESET COUNTER
INCREMENT COUNTER
LATCH NEXT INSTRUCTION GROUP
CONTROL SIGNALS

FIG._16

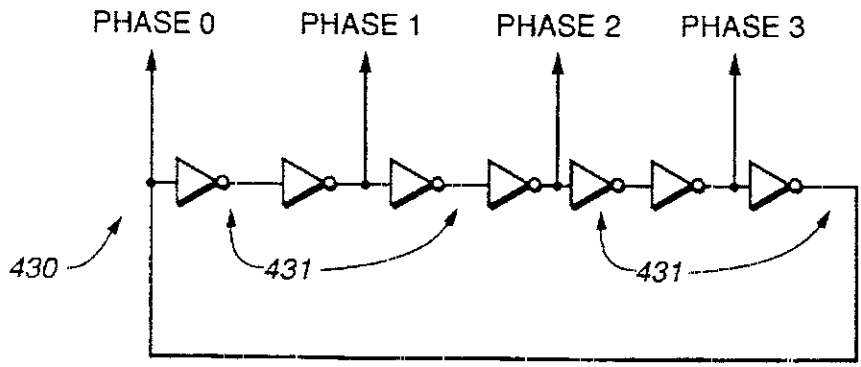

PHASE 0
PHASE 1
PHASE 2
PHASE 3
430
431
431

FIG._18

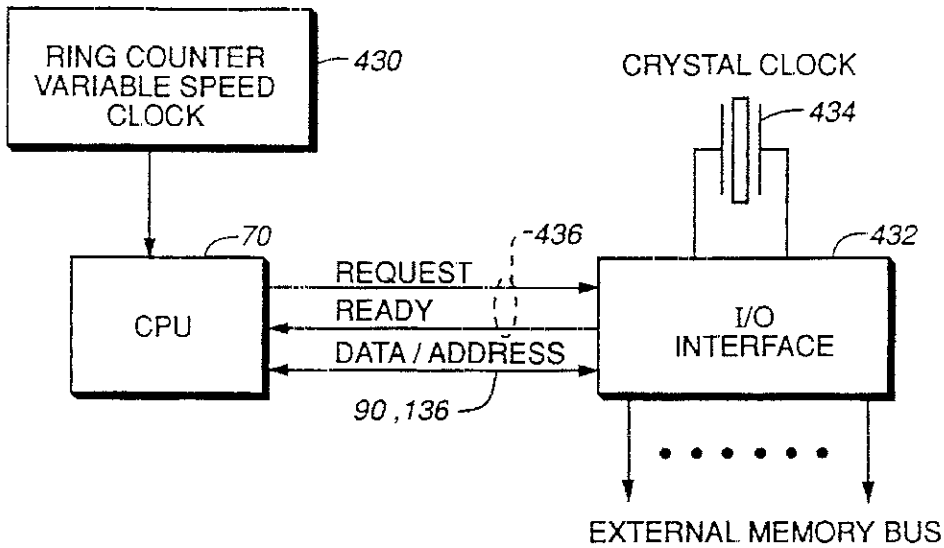

RING COUNTER
VARIABLE SPEED
CLOCK
430
CRYSTAL CLOCK
434
70
436
432
CPU
REQUEST
READY
DATA / ADDRESS
I/O
INTERFACE
90 ,136
EXTERNAL MEMORY BUS

FIG._17

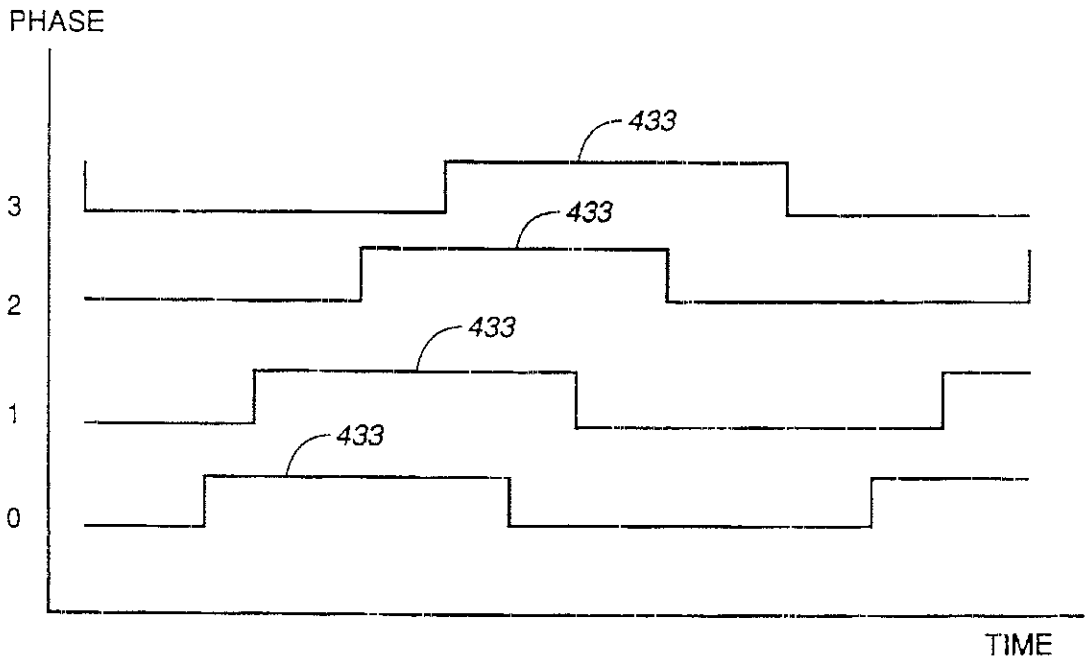

PHASE
433
433
433
433
3
2
1
0
TIME

FIG._19

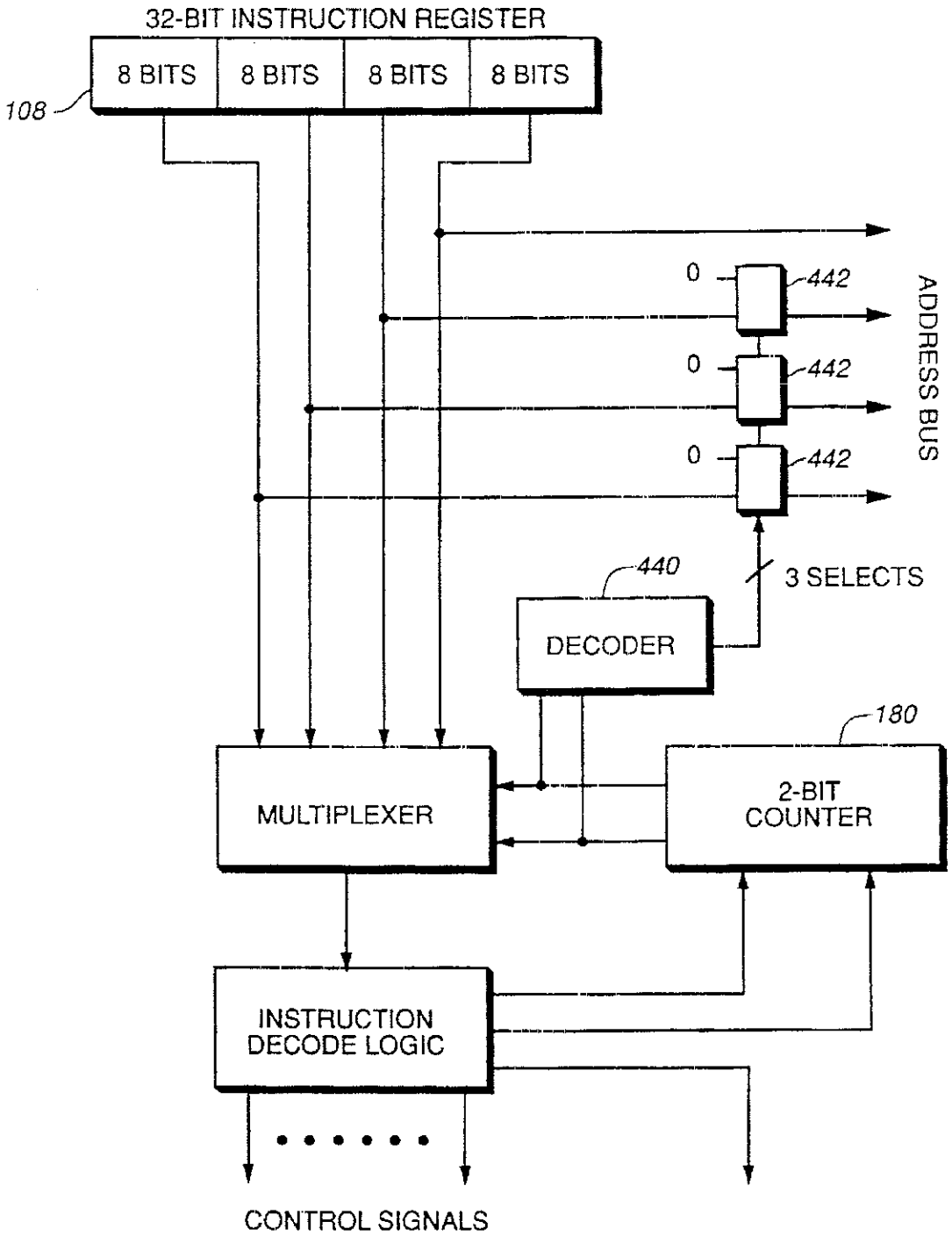

32-BIT INSTRUCTION REGISTER
8 BITS
8 BITS
8 BITS
8 BITS
108
0
442
0
442
0
442
ADDRESS BUS
440
3 SELECTS
DECODER
180
MULTIPLEXER
2-BIT COUNTER
INSTRUCTION DECODE LOGIC
CONTROL SIGNALS

FIG._20

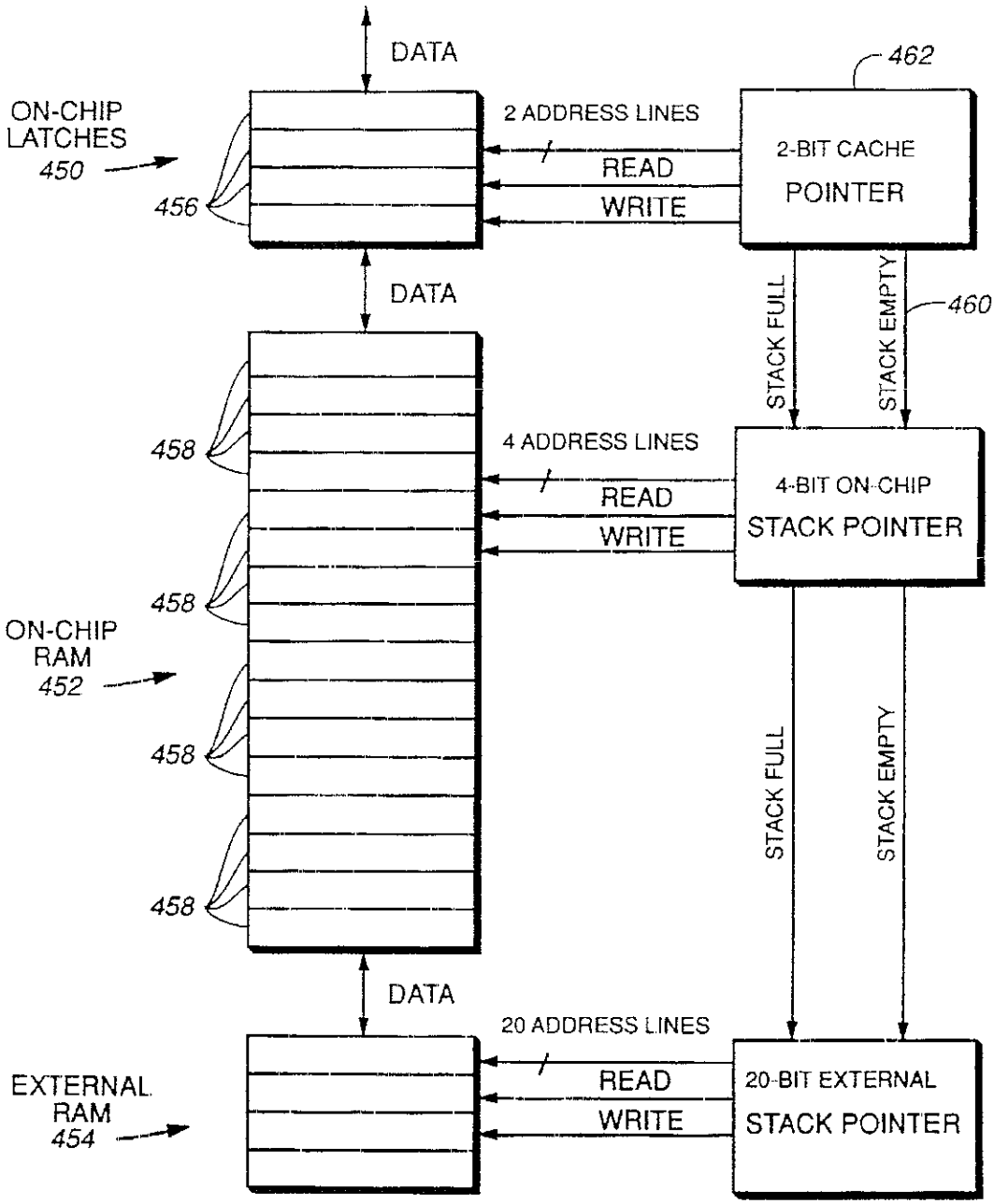

DATA
462
ON-CHIP
LATCHES
450
2 ADDRESS LINES
READ
WRITE
2-BIT CACHE
POINTER
456
DATA
STACK FULL
STACK EMPTY
460
458
4 ADDRESS LINES
READ
WRITE
4-BIT ON-CHIP
STACK POINTER
458
ON-CHIP
RAM
452
458
STACK FULL
STACK EMPTY
458
DATA
20 ADDRESS LINES
READ
WRITE
20-BIT EXTERNAL
STACK POINTER
EXTERNAL
RAM
454

FIG._21

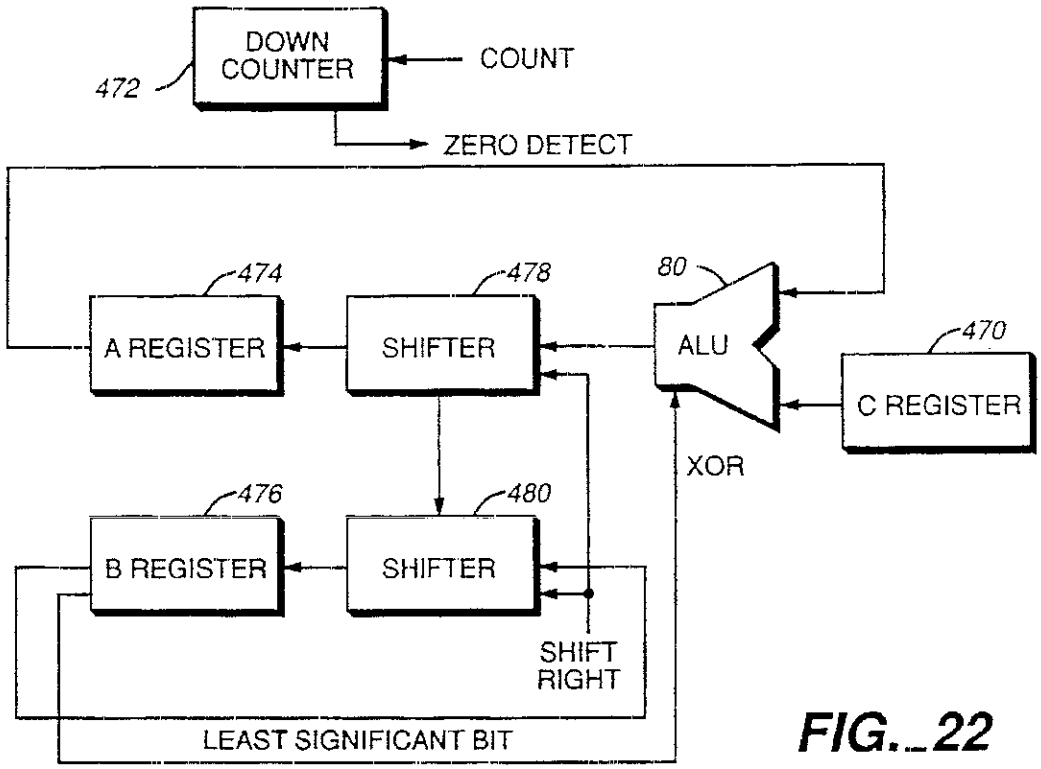

DOWN COUNTER
472
COUNT
ZERO DETECT
474
A REGISTER
478
SHIFTER
80
ALU
470
C REGISTER
XOR
476
B REGISTER
480
SHIFTER
SHIFT RIGHT
LEAST SIGNIFICANT BIT

FIG._22

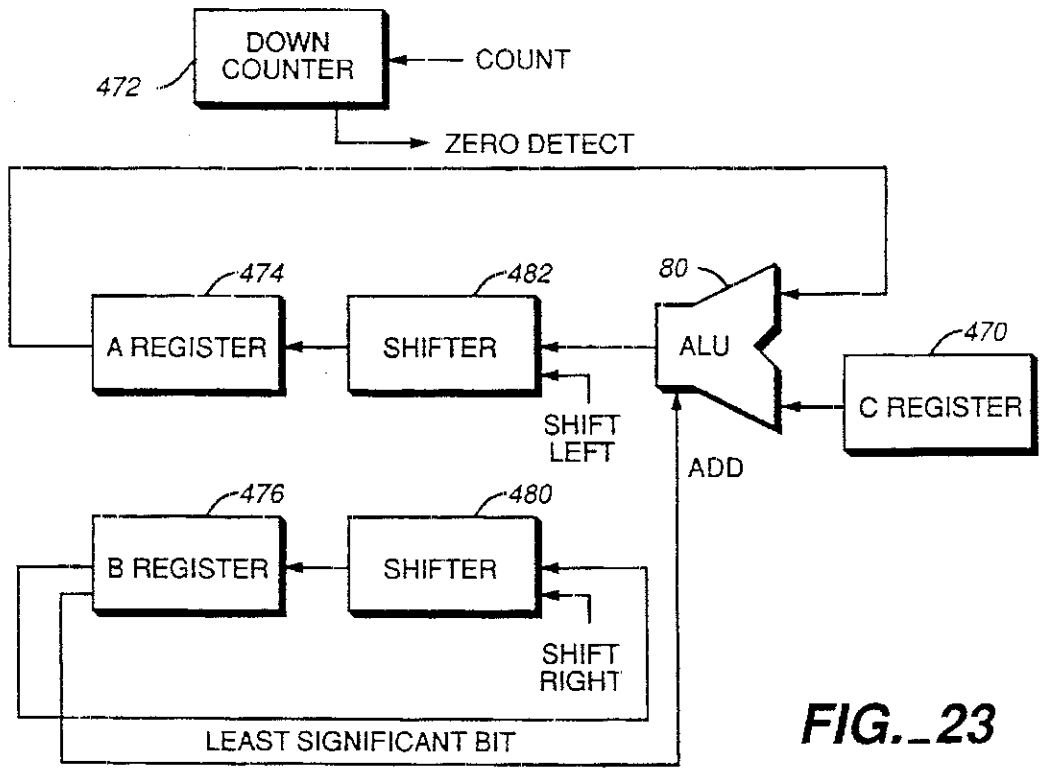

DOWN COUNTER
472
COUNT
ZERO DETECT
474
A REGISTER
482
SHIFTER
80
ALU
470
C REGISTER
SHIFT LEFT
ADD
476
B REGISTER
480
SHIFTER
SHIFT RIGHT
LEAST SIGNIFICANT BIT

FIG._23

HIGH PERFORMANCE, LOW COST MICROPROCESSOR

CROSS REFERENCE TO RELATED APPLICATIONS

This application is a division of U.S. application Ser. No. 07/389,334, filed Aug. 3, 1989 now U.S. Pat. No. 5,440,749

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to a simplified, reduced instruction set computer (RISC) microprocessor. More particularly, it relates to such a microprocessor which is capable of performance levels of, for example, 20 million instructions per second (MIPS) at a price of, for example, 20 dollars.

2. Description of the Prior Art

since the invention of the microprocessor, improvements in its design have taken two different approaches. In the first approach, a brute force gain in performance has been achieved through the provision of greater numbers of faster transistors in the microprocessor integrated circuit and an instruction set of increased complexity. This approach is exemplified by the Motorola 68000 and Intel 80X86 microprocessor families. The trend in this approach is to larger die sizes and packages, with hundreds of pinouts.

More recently, it has been perceived that performance gains can be achieved through comparative simplicity, both in the microprocessor integrated circuit itself and in its instruction set. This second approach provides RISC microprocessors, and is exemplified by the Sun SPARC and the Intel 8960 microprocessors. However, even with this approach as conventionally practiced, the packages for the microprocessor are large, in order to accommodate the large number of pinouts that continue to be employed. A need therefore remains for further simplification of high performance microprocessors.

With conventional high performance microprocessors, fast static memories are required for direct connection to the microprocessors in order to allow memory accesses that are fast enough to keep up with the microprocessors. Slower dynamic random access memories (DRAMs) are used with such microprocessors only in a hierarchical memory arrangement, with the static memories acting as a buffer between the microprocessors and the DRAMs. The necessity to use static memories increases cost of the resulting systems.

Conventional microprocessors provide direct memory accesses (DMA) for system peripheral units through DMA controllers, which may be located on the microprocessor integrated circuit, or provided separately. Such DMA controllers can provide routine handling of DMA requests and responses, but some processing by the main central processing unit (CPU) of the microprocessor is required.

SUMMARY OF THE INVENTION

Accordingly, it is an object of this invention to provide a microprocessor with a reduced pin count and cost compared to conventional microprocessors.

It is another object of the invention to provide a high performance microprocessor that can be directly connected to DRAMs without sacrificing microprocessor speed.

It is a further object of the invention to provide a high performance microprocessor in which DMA does not require use of the main CPU during DMA requests and responses and which provides very rapid DMA response with predictable response times.

The attainment of these and related objects may be achieved through use of the novel high performance, low cost microprocessor herein disclosed. In accordance with one aspect of the invention, a microprocessor system in accordance with this invention has a central processing unit, a dynamic random access memory and a bus connecting the central processing unit to the dynamic random access memory. There is a multiplexing means on the bus between the central processing unit and the dynamic random access memory. The multiplexing means is connected and configured to provide row addresses, column addresses and data on the bus.

In accordance with another aspect of the invention, the microprocessor system has a means connected to the bus for fetching instructions for the central processing unit on the bus. The means for fetching instructions is configured to fetch multiple sequential instructions in a single memory cycle. In a variation of this aspect of the invention, a programmable read only memory containing instructions for the central processing unit is connected to the bus. The means for fetching instructions includes means for assembling a plurality of instructions from the programmable read only memory and storing the plurality of instructions in the dynamic random access memory.

In another aspect of the invention, the microprocessor system includes a central processing unit, a direct memory access processing unit and a memory connected by a bus. The direct memory access processing unit includes means for fetching instructions for the central processing unit and for fetching instructions for the direct memory access processing unit on the bus.

In a further aspect of the invention, the microprocessor system, including the memory, is contained in an integrated circuit. The memory is a dynamic random access memory, and the means for fetching multiple instructions includes a column latch for receiving the multiple instructions.

In still another aspect of the invention, the microprocessor system additionally includes an instruction register for the multiple instructions connected to the means for fetching instructions. A means is connected to the instruction register for supplying the multiple instructions in succession from the instruction register. A counter is connected to control the means for supplying the multiple instructions to supply the multiple instructions in succession. A means for decoding the multiple instructions is connected to receive the multiple instructions in succession from the means for supplying the multiple instructions. The counter is connected to said means for decoding to receive incrementing and reset control signals from the means for decoding. The means for decoding is configured to supply the reset control signal to the counter and to supply a control signal to the means for fetching instructions in response to a SKIP instruction in the multiple instructions. In a modification of this aspect of the invention, the microprocessor system additionally has a loop counter connected to receive a decrement control signal from the means for decoding. The means for decoding is configured to supply the reset control signal to the counter and the decrement control signal to the loop counter in response to a MICROLOOP instruction in the multiple instructions. In a further modification to this aspect of the invention, the means for decoding is configured to control

the counter in response to an instruction utilizing a variable width operand. A means is connected to the counter to select the variable width operand in response to the counter

In a still further aspect of the invention, the microprocessor system includes an arithmetic logic unit. A first push down stack is connected to the arithmetic logic unit. The first push down stack includes means for storing a top item connected to a first input of the arithmetic logic unit and means for storing a next item connected to a second input of the arithmetic logic unit. The arithmetic logic unit has an output connected to the means for storing a top item. The means for storing a top item is connected to provide an input to a register file. The register file desirably is a second push down stack, and the means for storing a top item and the register file are bidirectionally connected

In another aspect of the invention, a data processing system has a microprocessor including a sensing circuit and a driver circuit, a memory, and an output enable line connected between the memory, the sensing circuit and the driver circuit. The sensing circuit is configured to provide a ready signal when the output enable line reaches a predetermined electrical level, such as a voltage. The microprocessor is configured so that the driver circuit provides an enabling signal on the output enable line responsive to the ready signal

In a further aspect of the invention, the microprocessor system has a ring counter variable speed system clock connected to the central processing unit. The central processing unit and the ring counter variable speed system clock are provided in a single integrated circuit. An input/output interface is connected to exchange coupling control signals, addresses and data with the input/output interface. A second clock independent of the ring counter variable speed system clock is connected to the input/output interface.

In yet another aspect of the invention, a push down stack is connected to the arithmetic logic unit. The push down stack includes means for storing a top item connected to a first input of the arithmetic logic unit and means for storing a next item connected to a second input of the arithmetic logic unit. The arithmetic logic unit has an output connected to the means for storing a top item. The push down stack has a first plurality of stack elements configured as latches and a second plurality of stack elements configured as a random access memory. The first and second plurality of stack elements and the central processing unit are provided in a single integrated circuit. A third plurality of stack elements is configured as a random access memory external to the single integrated circuit. In this aspect of the invention, desirably a first pointer is connected to the first plurality of stack elements, a second pointer connected to the second plurality of stack elements, and a third pointer is connected to the third plurality of stack elements. The central processing unit is connected to pop items from the first plurality of stack elements. The first stack pointer is connected to the second stack pointer to pop a first plurality of items from the second plurality of stack elements when the first plurality of stack elements are empty from successive pop operations by the central processing unit. The second stack pointer is connected to the third stack pointer to pop a second plurality of items from the third plurality of stack elements when the second plurality of stack elements are empty from successive pop operations by the central processing unit

In another aspect of the invention, a first register is connected to supply a first input to the arithmetic logic unit. A first shifter is connected between an output of the arithmetic logic unit and the first register. A second register is connected to receive a starting polynomial value. An output of the second register is connected to a second shifter. A least significant bit of the second register is connected to The arithmetic logic unit. A third register is connected to supply feedback terms of a polynomial to the arithmetic logic unit. A down counter, for counting down a number corresponding to digits of a polynomial to be generated, is connected to the arithmetic logic unit. The arithmetic logic unit is responsive to a polynomial instruction to carry out an exclusive OR of the contents of the first register with the contents of the third register if the least significant bit of the second register is a 'ONE" and to pass the contents of the first register unaltered if the least significant bit of the second register is a "ZERO", until the down counter completes a count. The polynomial to be generated results in said first register

In still another aspect of the invention, a result register is connected to supply a first input to the arithmetic logic unit. A first, left shifting shifter is connected between an output of the arithmetic logic unit and the result register. A multiplier register is connected to receive a multiplier in bit reversed form. An output of the multiplier register is connected to a second, right shifting shifter. A least significant bit of the multiplier register is connected to the arithmetic logic unit. A third register is connected to supply a multiplicand to said arithmetic logic unit. A down counter, for counting down a number corresponding to one less than the number of digits of the multiplier, is connected to the arithmetic logic unit. The arithmetic logic unit is responsive to a multiply instruction to add the contents of the result register with the contents of the third register, when the least significant bit of the multiplier register is a "ONE" and to pass the contents of the result register unaltered, until the down counter completes a count. The product results in the result register.

The attainment of the foregoing and related objects, advantages and features of the invention should be more readily apparent to those skilled in the art, after review of the following more detailed description of the invention, taken together with the drawings, in which:

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is an external, plan view of an integrated circuit package incorporating a microprocessor in accordance with the invention.

FIG. **2** is a block diagram of a microprocessor in accordance with the invention.

FIG. **3** is a block diagram of a portion of a data processing system incorporating the microprocessor of FIGS. **1** and **2**.

FIG. **4** is a more detailed block diagram of a portion of the microprocessor shown in FIG. **2**.

FIG. **5** is a more detailed block diagram of another portion of the microprocessor shown in FIG. **2**.

FIG. **6** is a block diagram of another portion of the data processing system shown in part in FIG. **3** and incorporating the microprocessor of FIGS. **1–2** and **4–5**.

FIGS. **7** and **8** are layout diagrams for the data processing system shown in part in FIGS. **3** and **6**.

FIG. **9** is a layout diagram of a second embodiment of a microprocessor in accordance with the invention in a data processing system on a single integrated circuit.

FIG. **10** is a more detailed block diagram of a portion of the data processing system of FIGS. **7** and **8**.

FIG. **11** is a timing diagram useful for understanding operation of the system portion shown in FIG. **12**.

FIG 12 is another more detailed block diagram of a further portion of the data processing system of FIGS 7 and 8

FIG 13 is a more detailed block diagram of a portion of the microprocessor shown in FIG 2

FIG 14 is a more detailed block and schematic diagram of a portion of the system shown in FIGS 3 and 7–8

FIG 15 is a graph useful for understanding operation of the system portion shown in FIG 14

FIG 16 is a more detailed block diagram showing part of the system portion shown in FIG 4

FIG 17 is a more detailed block diagram of a portion of the microprocessor shown in FIG 2

FIG 18 is a more detailed block diagram of part of the microprocessor portion shown in FIG 17

FIG 19 is a set of waveform diagrams useful for understanding operation of the part of the microprocessor portion shown in FIG 18

FIG 20 is a more detailed block diagram showing another part of the system portion shown in FIG 4

FIG 21 is a more detailed block diagram showing another part of the system portion shown in FIG 4

FIGS 22 and 23 are more detailed block diagrams showing another part of the system portion shown in FIG 4

DETAILED DESCRIPTION OF THE INVENTION

OVERVIEW

The microprocessor of this invention is desirably implemented as a 32-bit microprocessor optimized for:

HIGH EXECUTION SPEED and

LOW SYSTEM COST

In this embodiment, the microprocessor can be thought of as 20 MIPS for 20 dollars Important distinguishing features of the microprocessor are:

Uses low-cost commodity DYNAMIC RAMS to run 20 MIPS

4 instruction fetch per memory cycle

On-chip fast page-mode memory management

Runs fast without external cache

Requires few interfacing chips

Crams 32-bit CPU in 44 pin SOJ package

The instruction set is organized so that most operations can be specified with 8-bit instructions Two positive products of this philosophy are:

Programs are smaller,

Programs can execute much faster

The bottleneck in most computer systems is the memory bus The bus is used to fetch instructions and fetch and store data The ability to fetch four instructions in a single memory bus cycle significantly increases the bus availability to handle data

Turning now to the drawings, more particularly to FIG 1 there is shown a packaged 32-bit microprocessor **50** in a 44-pin plastic leadless chip carrier shown approximately 100 times its actual size of about 0 8 inch on a side The fact that the microprocessor **50** is provided as a 44-pin package represents a substantial departure from typical microprocessor packages, which usually have about 200 input/output (I/O) pins The microprocessor **50** is rated at 20 million instructions per second (MIPS) Address and data lines **52**, also labelled D0–D31, are shared for addresses and data without speed penalty as a result of the manner in which the microprocessor **50** operates, as will be explained below

DYNAMIC RAM

In addition to the low cost 44-pin package, another unusual aspect of the high performance microprocessor **50** is that it operates directly with dynamic random access memories (DRAMs), as shown by row address strobe (RAS) and column address strobe (CAS) I/O pins **54**. The other I/O pins for the microprocessor **50** include V_{DD} pins **56** V_{SS} pins **58**, output enable pin **60**, write pin **62**, clock pin **64** and reset pin **66**

All high speed computers require high speed and expensive memory to keep up The highest speed static RAM memories cost as much as ten times as much as slower dynamic RAMs. This microprocessor has been optimized to use low-cost dynamic RAM in high-speed page-mode. Page-mode dynamic RAMs offer static RAM performance without the cost penalty For example, low-cost 85 nsec. dynamic RAMs access at 25 nsec when operated in fast page-mode. Integrated fast page-mode control on the microprocessor chip simplifies system interfacing and results in a faster system

Details of the microprocessor **50** are shown in FIG 2 The microprocessor **50** includes a main central processing unit (CPU) **70** and a separate direct memory access (DMA) CPU **72** in a single integrated circuit making up the microprocessor **50** The main CPU **70** has a first 16 deep push down stack **74**, which has a top item register **76** and a next item register **78**, respectively connected to provide inputs to an arithmetic logic unit (ALU) **80** by lines **82** and **84** An output of the ALU **80** is connected to the top item register **76** by line **86**. The output of the top item register at **82** is also connected by line **88** to an internal data bus **90**

A loop counter **92** is connected to a decrementer **94** by lines **96** and **98**. The loop counter **92** is bidirectionally connected to the internal data bus **90** by line **100**. Stack pointer **102**, return stack pointer **104** mode register **106** and instruction register **108** are also connected to the internal data bus **90** by lines **110**, **112** **114** and **116**, respectively The internal data bus **90** is connected to memory controller **118** and to gate **120** The gate **120** provides inputs on lines **122**, **124**, and **126** to X register **128**, program counter **130** and Y register **132** of return push down stack **134** The X register **128**, program counter **130** and Y register **132** provide outputs to internal address bus **136** on lines **138**, **140** and **142** The internal address bus provides inputs to the memory controller **118** and to an incrementer **144** The incrementer **144** provides inputs to the X register, program counter and Y register via lines **146**, **122**, **124** and **126**. The DMA CPU **72** provides inputs to the memory controller **118** on line **148**. The memory controller **118** is connected to a RAM (not shown) by address/data bus **150** and control lines **152**

FIG 2 shows that the microprocessor **50** has a simple architecture Prior art RISC microprocessors are substantially more complex in design For example, the SPARC RISC microprocessor has three times the gates of the microprocessor **50** and the Intel 8960 RISC microprocessor has 20 times the gates of the microprocessor **50** The speed of this microprocessor is in substantial part due to this simplicity The architecture incorporates push down stacks and register write to achieve this simplicity.

The microprocessor **50** incorporates an I/O that has been tuned to make heavy use of resources provided on the integrated circuit chip On chip latches allow use of the same I/O circuits to handle three different things: column addressing, row addressing and data, with a slight to non-existent speed penalty. This triple bus multiplexing results in fewer buffers to expand fewer interconnection lines fewer I/O pins and fewer internal buffers.

The provision of on-chip DRAM control gives a performance equal to that obtained with the use of static RAMs As a result, memory is provided at ¼ the system cost of static RAM used in most RISC systems

The microprocessor **50** fetches 4 instructions per memory cycle; the instructions are in an 8-bit format, and this is a 32-bit microprocessor. System speed is therefore 4 times the memory bus bandwidth. This ability enables the microprocessor to break the Von Neumann bottleneck of the speed of getting the next instruction. This mode of operation is possible because of the use of a push down stack and register array. The push down stack allows the use of implied addresses, rather than the prior art technique of explicit addresses for two sources and a destination.

Most instructions execute in 20 nanoseconds in the microprocessor **50**. The microprocessor can therefore execute instructions at 50 peak MIPS without pipeline delays. This is a function of the small number of gates in the microprocessor **50** and the high degree of parallelism in the architecture of the microprocessor.

FIG. **3** shows how column and row addresses are multiplexed on lines D8–D14 of the microprocessor **50** for addressing DRAM **150** from I/O pins **52**. The DRAM **150** is one of eight, but only one DRAM **150** has been shown for clarity. As shown, the lines D11–D18 are respectively connected to row address inputs A0–A8 of the DRAM **150**. Additionally, lines D12–D15 are connected to the data inputs DQ1–DQ4 of the DRAM **150**. The output enable, write and column address strobe pins **54** are respectively connected to the output enable, write and column address strobe inputs of the DRAM **150** by lines **152**. The row address strobe pin **54** is connected through row address strobe decode logic **154** to the row address strobe input of the DRAM **150** by lines **156** and **158**.

D0–D7 pins **52** (FIG. **1**) are idle when the microprocessor **50** is outputting multiplexed row and column addresses on D11–D18 pins **52**. The D0–D7 pins **52** can therefore simultaneously be used for I/O when right justified I/O is desired. Simultaneous addressing and I/O can therefore be carried out.

FIG. **4** shows how the microprocessor **50** is able to achieve performance equal to the use of static RAMS with DRAMs through multiple instruction fetch in a single clock cycle and instruction fetch-ahead. Instruction register **108** receives four 8-bit byte instruction words **1–4** on 32-bit internal data bus **90**. The four instruction byte **1–4** locations of the instruction register **108** are connected to multiplexer **170** by busses **172**, **174**, **176** and **178**, respectively. A microprogram counter **180** is connected to the multiplexer **170** by lines **182**. The multiplexer **170** is connected to decoder **184** by bus **186**. The decoder **184** provides internal signals to the rest of the microprocessor **50** on lines **188**.

Most significant bits **190** of each instruction byte **1–4** location are connected to a 4-input decoder **192** by lines **194**. The output of decoder **192** is connected to memory controller **118** by line **196**. Program counter **130** is connected to memory controller **118** by internal address bus **136**, and the instruction register **108** is connected to the memory controller **118** by the internal data bus **90**. Address/data bus **198** and control bus **200** are connected to the DRAMS **150** (FIG. **3**).

In operation, when the most significant bits **190** of remaining instructions **1–4** are "1" in a clock cycle of the microprocessor **50**, there are no memory reference instructions in the queue. The output of decoder **192** on line **196** requests an instruction fetch ahead by memory controller **118** without interference with other accesses. While the current instructions in instruction register **108** are executing, the memory controller **118** obtains the address of the next set of four instructions from program counter **130** and obtains that set of instructions. By the time the current set of instructions has completed execution, the next set of instructions is ready for loading into the instruction register.

Details of the DMA CPU **72** are provided in FIG. **5**. Internal data bus **90** is connected to memory controller **118** and to DMA instruction register **210**. The DMA instruction register **210** is connected to DMA program counter **212** by bus **214**, to transfer size counter **216** by bus **218** and to timed transfer interval counter **220** by bus **222**. The DMA instruction register **210** is also connected to DMA I/O and RAM address register **224** by line **226**. The DMA I/O and RAM address register **224** is connected to the memory controller **118** by memory cycle request line **228** and bus **230**. The DMA program counter **212** is connected to the internal address bus **136** by bus **232**. The transfer size counter **216** is connected to a DMA instruction done decrementer **234** by lines **236** and **238**. The decrementer **234** receives a control input on memory cycle acknowledge line **240**. When transfer size counter **216** has completed its count, it provides a control signal to DMA program counter **212** on line **242**. Timed transfer interval counter **220** is connected to decrementer **244** by lines **246** and **248**. The decrementer **244** receives a control input from a microprocessor system clock on line **250**.

The DMA CPU **72** controls itself and has the ability to fetch and execute instructions. It operates as a co-processor to the main CPU **70** (FIG. **2**) for time specific processing.

FIG. **6** shows how the microprocessor **50** is connected to an electrically programmable read only memory (EPROM) **260** by reconfiguring the data lines **52** so that some of the data lines **52** are input lines and some of them are output lines. Data lines **52** D0–D7 provide data to and from corresponding data terminals **262** of the EPROM **260**. Data lines **52** D9–D18 provide addresses to address terminals **264** of the EPROM **260**. Data lines **52** D19–D31 provide inputs from the microprocessor **50** to memory and I/O decode logic **266**. RAS **0/1** control line **268** provides a control signal for determining whether the memory and I/O decode logic provides a DRAM RAS output on line **270** or a column enable output for the EPROM **260** on line **272**. Column address strobe terminal **60** of the microprocessor **50** provides an output enable signal on line **274** to the corresponding terminal **276** of the EPROM **260**.

FIGS. **7** and **8** show the front and back of a one card data processing system **280** incorporating the microprocessor **50**, MSM514258-10 type DRAMs **150** totalling 2 megabytes, a Motorola 50 MegaHertz crystal oscillator clock **282**, I/O circuits **284** and a 27256 type EPROM **260**. The I/O circuits **284** include a 74HC04 type high speed hex inverter circuit **286**, an IDT39C828 type 10-bit inverting buffer circuit **288**, an IDT39C822 type 10-bit inverting register circuit **290**, and two IDT39C823 type 9-bit non-inverting register circuits **292**. The card **280** is completed with a MAX12V type DC—DC converter circuit **294**, 34-pin dual AMP type headers **296**, a coaxial female power connector **298**, and a 3-pin AMP right angle header **300**. The card **280** is a low cost, imbeddable product that can be incorporated in larger systems or used as an internal development tool.

The microprocessor **50** is a very high performance (50 MHz) RISC influenced 32-bit CPU designed to work closely with dynamic RAM. Clock for clock, the microprocessor **50** approaches the theoretical performance limits possible with a single CPU configuration. Eventually, the microprocessor **50** and any other processor is limited by the bus bandwidth and the number of bus paths. The critical conduit is between the CPU and memory.

One solution to the bus bandwidth/bus path problem is to integrate a CPU directly onto the memory chips, giving every memory a direct bus the CPU. FIG. **9** shows another microprocessor **310** that is provided integrally with 1 mega-

bit of DRAM **311** in a single integrated circuit **312**. Until the present invention, this solution has not been practical, because most high performance CPUs require from 500,000 to 1,000,000 transistors and enormous die sizes just by themselves. The microprocessor **310** is equivalent to the microprocessor **50** in FIGS. **1–8**. The microprocessors **50** and **310** are the most transistor efficient high performance CPUs in existence, requiring fewer than 50,000 transistors for dual processors **70** and **72** (FIG. 2) or **314** and **316** (less memory). The very high speed of the microprocessors **50** and **310** is to a certain extent a function of the small number of active devices. In essence, the less silicon gets in the way, the faster the electrons can get where they are going.

The microprocessor **310** is therefore the only CPU suitable for integration on the memory chip die **312**. Some simple modifications to the basic microprocessor **50** to take advantage of the proximity to the DRAM array **311** can also increase the microprocessor **50** clock speed by 50 percent and probably more.

The microprocessor **310** core on board the DRAM die **312** provides most of the speed and functionality required for a large group of applications from automotive to peripheral control. However the integrated CPU **310**/DRAM **311** concept has the potential to redefine significantly the way multiprocessor solutions can solve a spectrum of very compute intensive problems. The CPU **310**/DRAM **311** combination eliminates the Von Neumann bottleneck by distributing it across numerous CPU/DRAM chips **312**. The microprocessor **310** is a particularly good core for multiprocessing, since it was designed with the SDI targeting array in mind, and provisions were made for efficient interprocessor communications.

Traditional multiprocessor implementations have been very expensive in addition to being unable to exploit fully the available CPU horsepower. Multiprocessor systems have typically been built up from numerous board level or box level computers. The result is usually an immense amount of hardware with corresponding wiring, power consumption and communications problems. By the time the systems are interconnected as much as 50 percent of the bus speed has been utilized just getting through the interfaces.

In addition, multiprocessor system software has been scarce. A multiprocessor system can easily be crippled by an inadequate load-sharing algorithm in the system software, which allows one CPU to do a great deal of work and the others to be idle. Great strides have been made recently in systems software, and even UNIX V4 may be enhanced to support multiprocessing. Several commercial products from such manufacturers as DUAL Systems and UNISOFT do a credible job on 68030 type microprocessor systems now.

The microprocessor **310** architecture eliminates most of the interface friction, since up to 64 CPU **310**/RAM **311** processors should be able to intercommunicate without buffers or latches. Each chip **312** has about 40 MIPS raw speed, because placing the DRAM **311** next to the CPU **310** allows the microprocessor **310** instruction cycle to be cut in half, compared to the microprocessor **50**. A 64 chip array of these chips **312** is more powerful than any other existing computer. Such an array fits on a 3×5 card, cost less than a FAX machine, and draw about the same power as a small television.

Dramatic changes in price/performance always reshape existing applications and almost always create new ones. The introduction of microprocessors in the mid 1970s created video games, personal computers, automotive computers, electronically controlled appliances, and low cost computer peripherals.

The integrated circuit **312** will find applications in all of the above areas, plus create some new ones. A common generic parallel processing algorithm handles convolution/Fast Fourier Transform (FFT)/pattern recognition. Interesting product possibilities using the integrated circuit **312** include high speed reading machines, real-time speech recognition, spoken language translation, real-time robot vision, a product to identify people by their faces, and an automotive or aviation collision avoidance system.

A real time processor for enhancing high density television (HDTV) images, or compressing the HDTV information into a smaller bandwidth, would be very feasible. The load sharing in HDTV could be very straightforward. Splitting up the task according to color and frame would require 6, 9 or 12 processors. Practical implementation might require 4 meg RAMs integrated with the microprocessor **310**.

The microprocessor **310** has the following specifications:

CONTROL LINES
4 - POWER/GROUND
1 - CLOCK
32 - DATA I/O
4 - SYSTEM CONTROL
  EXTERNAL MEMORY FETCH
  EXTERNAL MEMORY FETCH AUTOINCREMENT X
  EXTERNAL MEMORY FETCH AUTOINCREMENT Y
  EXTERNAL MEMORY WRITE
  EXTERNAL MEMORY WRITE AUTOINCREMENT X
  EXTERNAL MEMORY WRITE AUTOINCREMENT Y
  EXTERNAL PROM FETCH
  LOAD ALL X REGISTERS
  LOAD ALL Y REGISTERS
  LOAD ALL PC REGISTERS
  EXCHANGE X AND Y
  INSTRUCTION FETCH
  ADD TO PC
  ADD TO X
  WRITE MAPPING REGISTER
  READ MAPPING REGISTER
REGISTER CONFIGURATION
MICROPROCESSOR **310** CPU **316** CORE
COLUMN LATCH1 (1024 BITS) 32×32 MUX
STACK POINTER (16 BITS)
COLUMN LATCH2 (1024 BITS) 32×32 MUX
RSTACK POINTER (16 BITS)
PROGRAM COUNTER 32 BITS
X0 REGISTER 32 BITS (ACTIVATED ONLY FOR ON-CHIP ACCESSES)
Y0 REGISTER 32 BITS (ACTIVATED ONLY FOR ON-CHIP ACCESSES)
LOOP COUNTER 32 BITS
DMA CPU **314** CORE
DMA PROGRAM COUNTER 24 BITS
INSTRUCTION REGISTER 32 BITS
I/O & RAM ADDRESS REGISTER 32 BITS
TRANSFER SIZE COUNTER 12 BITS
INTERVAL COUNTER 12 BITS

To offer memory expansion for the basic chip **312**, an intelligent DRAM can be produced. This chip will be optimized for high speed operation with the integrated circuit **312** by having three on-chip address registers: Program Counter, X Register and Y register. As a result, to access the intelligent DRAM, no address is required and a total access cycle could be as short as 10 nsec. Each expansion DRAM would maintain its own copy of the three registers and would be identified by a code specifying its memory address. Incrementing and adding to the three

registers will actually take place on the memory chips. A maximum of 64 intelligent DRAM peripherals would allow a large system to be created without sacrificing speed by introducing multiplexers or buffers

There are certain differences between the microprocessor **310** and the microprocessor **50** that arise from providing the microprocessor **310** on the same die **312** with the DRAM **311**. Integrating the DRAM **311** allows architectural changes in the microprocessor **310** logic to take advantage of existing on-chip DRAM **311** circuitry. Row and column design is inherent in memory architecture. The DRAMs **311** access random bits in a memory array by first selecting a row of 1024 bits, storing them into a column latch, and then selecting one of the bits as the data to be read or written.

The time required to access the data is split between the row access and the column access. Selecting data already stored in a column latch is faster than selecting a random bit by at least a factor of six. The microprocessor **310** takes advantage of this high speed by creating a number of column latches and using them as caches and shift registers. Selecting a new row of information may be thought of as performing a 1024-bit read or write with the resulting immense bus bandwidth.

1. The microprocessor **50** treats its 32-bit instruction register **108** (see FIGS. **2** and **4**) as a cache for four 8-bit instructions. Since the DRAM **311** maintains a 1024-bit latch for the column bits, the microprocessor **310** treats the column latch as a cache for 128 8-bit instructions. Therefore, the next instruction will almost always be already present in the cache. Long loops within the cache are also possible and more useful than the 4 instruction loops in the microprocessor **50**.

2. The microprocessor **50** uses two 16×32-bit deep register arrays **74** and **134** (FIG. **2**) for the parameter stack and the return stack. The microprocessor **310** creates two other 1024-bit column latches to provide the equivalent of two 32×32-bit arrays, which can be accessed twice as fast as a register array.

3. The microprocessor **50** has a DMA capability which can be used for I/O to a video shift register. The microprocessor **310** uses yet another 1024-bit column latch as a long video shift register to drive a CRT display directly. For color displays, three on-chip shift registers could also be used. These shift registers can transfer pixels at a maximum of 100 MHz.

4. The microprocessor **50** accesses memory via an external 32-bit bus. Most of the memory **311** for the microprocessor **310** is on the same die **312**. External access to more memory is made using an 8-bit bus. The result is a smaller die, smaller package and lower power consumption than the microprocessor **50**.

5. The microprocessor **50** consumes about a third of its operating power charging and discharging the I/O pins and associated capacitances. The DRAMs **150** (FIG. **8**) connected to the microprocessor **50** dissipate most of their power in the I/O drivers. A microprocessor **310** system will consume about one-tenth the power of a microprocessor **50** system, since having the DRAM **311** next to the processor **310** eliminates most of the external capacitances to be charged and discharged.

6. Multiprocessing means splitting a computing task between numerous processors in order to speed up the solution. The popularity of multiprocessing is limited by the expense of current individual processors as well as the limited interprocessor communications ability. The microprocessor **310** is an excellent multiprocessor candidate since the chip **312** is a monolithic computer complete with memory, rendering it low-cost and physically compact.

The shift registers implemented with the microprocessor **310** to perform video output can also be configured as interprocessor communication links. The INMOS transputer attempted a similar strategy, but at much lower speed and without the performance benefits inherent in the microprocessor **310** column latch architecture. Serial I/O is a prerequisite for many multiprocessor topologies because of the many neighbor processors which communicate. A cube has 6 neighbors. Each neighbor communicates using these lines:

DATA IN
CLOCK IN
READY FOR DATA
DATA OUT
DATA READY?
CLOCK OUT

A special start up sequence is used to initialize the on-chip DRAM **311** in each of the processors.

The microprocessor **310** column latch architecture allows neighbor processors to deliver information directly to internal registers or even instruction caches of other chips **312**. This technique is not used with existing processors, because it only improves performance in a tightly coupled DRAM system.

7. The microprocessor **50** architecture offers two types of looping structures: LOOP-IF-DONE and MICRO-LOOP. The former takes an 8-bit to 24-bit operand to describe the entry point to the loop address. The latter performs a loop entirely within the 4 instruction queue and the loop entry point is implied as the first instruction in the queue. Loops entirely within the queue run without external instruction fetches and execute up to three times as fast as the long loop construct. The microprocessor **310** retains both constructs with a few differences. The microprocessor **310** microloop functions in the same fashion as the microprocessor **50** operation except the queue is 1024-bits or 128 8-bit instructions long. The microprocessor **310** microloop can therefore contain jumps, branches, calls and immediate operations not possible in the 4 8-bit instruction microprocessor **50** queue.

Microloops in the microprocessor **50** can only perform simple block move and compare functions. The larger microprocessor **310** queue allows entire digital signal processing or floating point algorithms to loop at high speed in the queue.

The microprocessor **50** offers four instructions to redirect execution:

CALL
BRANCH
BRANCH-IF-ZERO
LOOP-IF-NOT-DONE

These instructions take a variable length address operand 8, 16 or 24 bits long. The microprocessor **50** next address logic treats the three operands similarly by adding or subtracting them to the current program counter. For the microprocessor **310**, the 16 and 24-bit operands function in the same manner as the 16 and 24-bit operands in the microprocessor **50**. The 8-bit class operands are reserved to operate entirely within the instruction queue. Next address decisions can therefore be made quickly because only 10 bits of addresses are affected, rather than 32. There is no carry or borrow generated past the 10 bits.

8. The microprocessor **310** CPU **316** resides on an already crowded DRAM die **312**. To keep chip size as small as possible, the DMA processor **72** of the microprocessor **50** has been replaced with a more traditional DMA controller **314**. DMA is used with the microprocessor **310** to perform the following functions:

Video output to a CRT

Multiprocessor serial communications
8-bit parallel I/O

The DMA controller **314** can maintain both serial and parallel transfers simultaneously. The following DMA sources and destinations are supported by the microprocessor **310**:

DESCRIPTION	I/O	LINES
1 Video shift register	OUTPUT	1 to 3
2 Multiprocessor serial	BOTH	6 lines/channel
3 8-bit parallel	BOTH	8 data 4 control

The three sources use separate 1024-bit buffers and separate I/O pins. Therefore, all three may be active simultaneously without interference.

The microprocessor **310** can be implemented with either a single multiprocessor serial buffer or separate receive and sending buffers for each channel, allowing simultaneous bidirectional communications with six neighbors simultaneously.

FIGS. **10** and **11** provide details of the PROM DMA used in the microprocessor **50**. The microprocessor **50** executes faster than all but the fastest PROMs. PROMS are used in a microprocessor **50** system to store program segments and perhaps entire programs. The microprocessor **50** provides a feature on power-up to allow programs to be loaded from low-cost, slow speed PROMs into high speed DRAM for execution. The logic which performs this function is part of the DMA memory controller **118**. The operation is similar to DMA, but not identical, since four 8-bit bytes must be assembled on the microprocessor **50** chip, then written to the DRAM **150**.

The microprocessor **50** directly interfaces to DRAM **150** over a triple multiplexed data and address bus **350**, which carries RAS addresses, CAS addresses and data. The EPROM **260**, on the other hand, is read with non-multiplexed busses. The microprocessor **50** therefore has a special mode which unmultiplexes the data and address lines to read 8 bits of EPROM data. Four 8-bit bytes are read in this fashion. The multiplexed bus **350** is turned back on, and the data is written to the DRAM **150**.

When the microprocessor **50** detects a RESET condition, the processor stops the main CPU **70** and forces a mode **0** (PROM LOAD) instruction into the DMA CPU **72** instruction register. The DMA instruction directs the memory controller to read the EPROM **260** data at 8 times the normal access time for memory. Assuming a 50 MHz microprocessor **50**, this means an access time of 320 nsec. The instruction also indicates:

The selection address of the EPROM **260** to be loaded.
The number of 32-bit words to transfer.
The DRAM **150** address to transfer into.

The sequence of activities to transfer one 32-bit word from EPROM **260** to DRAM **150** are:

1. RAS goes low at **352**, latching the EPROM **260** select information from the high order address bits. The EPROM **260** is selected.
2. Twelve address bits (consisting of what is normally DRAM CAS addresses plus two byte select bits are placed on the bus **350** going to the EPROM **260** address pins. These signals will remain on the lines until the data from the EPROM **260** has been read into the microprocessor **50**. For the first byte, the byte select bits will be binary 00.
3. CAS goes low at **354**, enabling the EPROM **260** data onto the lower 8 bits of the external address/data bus **350**. NOTE: It is important to recognize that, during this part of the cycle, the lower 8 bits of the external data/address bus are functioning as inputs, but the rest of the bus is still acting as outputs.
4. The microprocessor **50** latches these eight least significant bits internally and shifts them 8 bits left to shift them to the next significant byte position.
5. Steps **2**, **3** and **4** are repeated with byte address **01**.
6. Steps **2**, **3** and **4** are repeated with byte address **10**.
7. Steps **2**, **3** and **4** are repeated with byte address **11**.
8. CAS goes high at **356**, taking the EPROM **260** off the data bus.
9. RAS goes high at **358**, indicating the end of the EPROM **260** access.
10. RAS goes low at **360**, latching the DRAM select information from the high order address bits. At the same time, the RAS address bits are latched into the DRAM **150**. The DRAM **150** is selected.
11. CAS goes low at **362**, latching the DRAM **150** CAS addresses.
12. The microprocessor **50** places the previously latched EPROM **260** 32-bit data onto the external address/data bus **350**. W goes low at **364**, writing the 32 bits into the DRAM **150**.
13. W goes high at **366**. CAS goes high at **368**. The process continues with the next word.

FIG. **12** shows details of the microprocessor **50** memory controller **118**. In operation, bus requests stay present until they are serviced. CPU **70** requests are prioritized at **370** in the order of: 1, Parameter Stack; 2, Return Stack; 3, Data Fetch; 4, Instruction Fetch. The resulting CPU request signal and a DMA request signal are supplied as bus requests to bus control **372**, which provides a bus grant signal at **374**. Internal address bus **136** and a DMA counter **376** provide inputs to a multiplexer **378**. Either a row address or a column address are provided as an output to multiplexed address bus **380** as an output from the multiplexer **378**. The multiplexed address bus **380** and the internal data bus **90** provide address and data inputs, respectively, to multiplexer **382**. Shift register **384** supplies row address strobe (RAS) **1** and **2** control signals to multiplexer **386** and column address strobe (CAS) **1** and **2** control signals to multiplexer **388** on lines **390** and **392**. The shift register **384** also supplies output enable (OE) and write (W) signals on lines **394** and **396** and a control signal on line **398** to multiplexer **382**. The shift register **384** receives a RUN signal on line **400** to generate a memory cycle and supplies a MEMORY READY signal on line **402** when an access is complete.

STACK/REGISTER ARCHITECTURE

Most microprocessors use on-chip registers for temporary storage of variables. The on-chip registers access data faster than off-chip RAM. A few microprocessors use an on-chip push down stack for temporary storage.

A stack has the advantage of faster operation compared to on-chip registers by avoiding the necessity to select source and destination registers. (A math or logic operation always uses the top two stack items as source and the top of stack as destination.) The stack's disadvantage is that it makes some operations clumsy. Some compiler activities in particular require on-chip registers for efficiency.

As shown in FIG. **13**, the microprocessor **50** provides both on-chip registers **134** and a stack **74** and reaps the benefits of both.

BENEFITS:

1. Stack math and logic is twice as fast as those available on an equivalent register only machine. Most program-

mers and optimizing compilers can take advantage of this feature.

2. Sixteen registers are available for on-chip storage of local variables which can transfer to the stack for computation. The accessing of variables is three to four times as fast as available on a strictly stack machine

The combined stack **74**/register **134** architecture has not been used previously due to inadequate understanding by computer designers of optimizing compilers and the mix of transfer versus math/logic instructions.

ADAPTIVE MEMORY CONTROLLER

A microprocessor must be designed to work with small or large memory configurations. As more memory loads are added to the data, address, and control lines, the switching speed of the signals slows down. The microprocessor **50** multiplexes the address/data bus three ways, so timing between the phases is critical. A traditional approach to the problem allocates a wide margin of time between bus phases so that systems will work with small or large numbers of memory chips connected. A speed compromise of as much as 50% is required.

As shown in FIG. **14**, the microprocessor **50** uses a feedback technique to allow the processor to adjust memory bus timing to be fast with small loads and slower with large ones. The OUTPUT ENABLE (OE) line **152** from the microprocessor **50** is connected to all memories **150** on the circuit board. The loading on the output enable line **152** to the microprocessor **50** is directly related to the number of memories **150** connected. By monitoring how rapidly OE **152** goes high after a read, the microprocessor **50** is able to determine when the data hold time has been satisfied and place the next address on the bus.

The level of the OE line **152** is monitored by CMOS input buffer **410** which generates an internal READY signal on line **412** to the microprocessor's memory controller. Curves **414** and **416** of the FIG. **15** graph show the difference in rise time likely to be encountered from a lightly to heavily loaded memory system. When the OE line **152** has reached a predetermined level to generate the READY signal, driver **418** generates an OUTPUT ENABLE signal on OE line **152**.

SKIP WITHIN THE INSTRUCTION CACHE

The microprocessor **50** fetches four 8-bit instructions each memory cycle and stores them in a 32-bit instruction register **108**, as shown in FIG. **16**. A class of "test and skip" instructions can very rapidly execute a very fast jump operation within the four instruction cache.

SKIP CONDITIONS:

Always

ACC non-zero

ACC negative

Carry flag equal logic one

Never

ACC equal zero

ACC positive

Carry flag equal logic zero

The SKIP instruction can be located in any of the four byte positions **420** in the 32-bit instruction register **108**. If the test is successful, SKIP will jump over the remaining one, two, or three 8-bit instructions in the instruction register **108** and cause the next four-instruction group to be loaded into the register **108**. As shown, the SKIP operation is implemented by resetting the 2-bit microinstruction counter **180** to zero on line **422** and simultaneously latching the next instruction group into the register **108**. Any instructions following the SKIP in the instruction register are overwritten by the new instructions and not executed.

The advantage of SKIP is that optimizing compilers and smart programmers can often use it in place of the longer conditional JUMP instruction. SKIP also makes possible microloops which exit when the loop counts down or when the SKIP jumps to the next instruction group. The result is very fast code.

Other machines (such as the PDP-**8** and Data General NOVA) provide the ability to skip a single instruction. The microprocessor **50** provides the ability to skip up to three instructions.

MICROLOOP IN THE INSTRUCTION CACHE

The microprocessor **50** provides the MICROLOOP instruction to execute repetitively from one to three instructions residing in the instruction register **108**. The microloop instruction works in conjunction with the LOOP COUNTER **92** (FIG. **2**) connected to the internal data bus **90**. To execute a microloop, the program stores a count in LOOP COUNTER **92**. MICROLOOP may be placed in the first, second, third, or last byte **420** of the instruction register **108**. If placed in the first position, execution will just create a delay equal to the number stored in LOOP COUNTER **92** times the machine cycle. If placed in the second, third, or last byte **420**, when the microloop instruction is executed, it will test the LOOP COUNT for zero. If zero, execution will continue with the next instruction. If not zero, the LOOP COUNTER **92** is decremented and the 2-bit microinstruction counter is cleared, causing the preceding instructions in the instruction register to be executed again.

Microloop is useful for block move and search operations. By executing a block move completely out of the instruction register **108**, the speed of the move is doubled, since all memory cycles are used by the move rather than being shared with instruction fetching. Such a hardware implementation of microloops is much faster than conventional software implementation of a comparable function.

OPTIMAL CPU CLOCK SCHEME

The designer of a high speed microprocessor must produce a product which operate over wide temperature ranges, wide voltage swings, and wide variations in semiconductor processing. Temperature, voltage, and process all affect transistor propagation delays. Traditional CPU designs are done so that with the worse case of the three parameters the circuit will function at the rated clock speed. The result are designs that must be clocked a factor of two slower than their maximum theoretical performance, so they will operate properly in worse case conditions.

The microprocessor **50** uses the technique shown in FIGS **17–19** to generate the system clock and its required phases. Clock circuit **430** is the familiar "ring oscillator" used to test process performance. The clock is fabricated on the same silicon chip as the rest of the microprocessor **50**.

The ring oscillator frequency is determined by the parameters of temperature, voltage, and process. At room temperature, the frequency will be in the neighborhood of 100 MHZ. At 70 degrees Centigrade, the speed will be 50 MHZ. The ring oscillator **430** is useful as a system clock, with its stages **431** producing phase 0-phase **3** outputs **433** shown in FIG. **19**, because its performance tracks the parameters which similarly affect all other transistors on the same silicon die. By deriving system timing from the ring oscillator **430**, CPU **70** will always execute at the maximum frequency possible, but never too fast. For example, if the processing of a particular die is not good resulting in slow transistors, the latches and gates on the microprocessor **50** will operate slower than normal. Since the microprocessor **50** ring oscillator clock **430** is made from the same transistors on the same die as the latches and gates, it too will

operate slower (oscillating at a lower frequency) providing compensation which allows the rest of the chip s logic to operate properly.

ASYNCHRONOUS/SYNCHRONOUS CPU

Most microprocessors derive all system timing from a single clock. The disadvantage is that different parts of the system can slow all operations. The microprocessor **50** provides a dual-clock scheme as shown in FIG. **17**, with the CPU **70** operating asynchronously to I/O interface **432** forming part of memory controller **118** (FIG. **2**) and the I/O interface **432** operating synchronously with the external world of memory and I/O devices. The CPU **70** executes at the fastest speed possible using the adaptive ring oscillator clock **430**. Speed may vary by a factor of four depending upon temperature, voltage, and process. The external world must be synchronized to the microprocessor **50** for operations such as video display updating and disc drive reading and writing. This synchronization is performed by the I/O interface **432**, speed of which is controlled by a conventional crystal clock **434**. The interface **432** processes requests for memory accesses from the microprocessor **50** and acknowledges the presence of I/O data. The microprocessor **50** fetches up to four instructions in a single memory cycle and can perform much useful work before requiring another memory access. By decoupling the variable speed of the CPU **70** from the fixed speed of the I/O interface **432**, optimum performance can be achieved by each. Recoupling between the CPU **70** and the interface **432** is accomplished with hand shake signals on lines **436**, with data/addresses passing on bus **90**, **136**.

ASYNCHRONOUS/SYNCHRONOUS CPU IMBEDDED ON A DRAM CHIP

System performance is enhanced even more when the DRAM **311** and CPU **314** (FIG. **9**) are located on the same die. The proximity of the transistors means that DRAM **311** and CPU **314** parameters will closely follow each other. At room temperature, not only would the CPU **314** execute at 100 MHZ, but the DRAM **311** would access fast enough to keep up. The synchronization performed by the I/O interface **432** would be for DMA and reading and writing I/O ports. In some systems (such as calculators) no I/O synchronization at all would be required, and the I/O clock would be tied to the ring counter clock.

VARIABLE WIDTH OPERANDS

Many microprocessors provide variable width operands. The microprocessor **50** handles operands of 8, 16, or 24 bits using the same op-code. FIG. **20** shows the 32-bit instruction register **108** and the 2-bit microinstruction register **180** which selects the 8-bit instruction. Two classes of microprocessor **50** instructions can be greater than 8-bits, JUMP class and IMMEDIATE. A JUMP or IMMEDIATE op-code is 8-bits, but the operand can be 8, 16, or 24 bits long. This magic is possible because operands must be right justified in the instruction register. This means that the least significant bit of the operand is always located in the least significant bit of the instruction register. The microinstruction counter **180** selects which 8-bit instruction to execute. If a JUMP or IMMEDIATE instruction is decoded, the state of the 2-bit microinstruction counter selects the required 8, 16, or 24 bit operand onto the address or data bus. The unselected 8-bit bytes are loaded with zeros by operation of decoder **440** and gates **442**. The advantage of this technique is the saving of a number of op-codes required to specify the different operand sizes in other microprocessors.

TRIPLE STACK CACHE

Computer performance is directly related to the system memory bandwidth. The faster the memories, the faster the computer. Fast memories are expensive, so techniques have been developed to move a small amount of high-speed memory around to the memory addresses where it is needed. A large amount of slow memory is constantly updated by the fast memory, giving the appearance of a large fast memory array. A common implementation of the technique is known as a high-speed memory cache. The cache may be thought of as fast acting shock absorber smoothing out the bumps in memory access. When more memory is required than the shock can absorb, it bottoms out and slow speed memory is accessed. Most memory operations can be handled by the shock absorber itself.

The microprocessor **50** architecture has the ALU **80** (FIG. **2**) directly coupled to the top two stack locations **76** and **78**. The access time of the stack **74** therefore directly affects the execution speed of the processor. The microprocessor **50** stack architecture is particularly suitable to a triple cache technique, shown in FIG. **21** which offers the appearance of a large stack memory operating at the speed of on-chip latches **450**. Latches **450** are the fastest form of memory device built on the chip, delivering data in as little as 3 nsec. However latches **450** require large numbers of transistors to construct. On-chip RAM **452** requires fewer transistors than latches, but is slower by a factor of five (15 nsec access). Off-chip RAM **150** is the slowest storage of all. The microprocessor **50** organizes the stack memory hierarchy as three interconnected stacks **450**, **452** and **454**. The latch stack **450** is the fastest and most frequently used. The on-chip RAM stack **452** is next. The off-chip RAM stack **454** is slowest. The stack modulation determines the effective access time of the stack. If a group of stack operations never push or pull more than four consecutive items on the stack, operations will be entirely performed in the 3 nsec latch stack. When the four latches **456** are filled, the data in the bottom of the latch stack **450** is written to the top of the on-chip RAM stack **452**. When the sixteen locations **458** in the on-chip RAM stack **452** are filled, the data in the bottom of the on-chip RAM stack **452** is written to the top of the off-chip RAM stack **454**. When popping data off a full stack **450**, four pops will be performed before stack empty line **460** from the latch stack pointer **462** transfers data from the on-chip RAM stack **452**. By waiting for the latch stack **450** to empty before performing the slower on-chip RAM access, the high effective speed of the latches **456** are made available to the processor. The same approach is employed with the on-chip RAM stack **452** and the off-chip RAM stack **454**.

POLYNOMIAL GENERATION INSTRUCTION

Polynomials are useful for error correction, encryption, data compression, and fractal generation. A polynomial is generated by a sequence of shift and exclusive OR operations. Special chips are provided for this purpose in the prior art.

The microprocessor **50** is able to generate polynomials at high speed without external hardware by slightly modifying how the ALU **80** works. As shown in FIG. **21**, a polynomial is generated by loading the 'order' (also known as the feedback terms) into C Register **470**. The value thirty one (resulting in 32 iterations) is loaded into DOWN COUNTER **472**. A register **474** is loaded with zero. B register **476** is loaded with the starting polynomial value. When the POLY instruction executes, C register **470** is exclusively ORed with A register **474** if the least significant bit of B register **476** is a one. Otherwise, the contents of the A register **474** passes through the ALU **80** unaltered. The combination of A and B is then shifted right (divided by 2) with shifters **478** and **480**. The operation automatically repeats the specified number of iterations, and the resulting polynomial is left in A register **474**.

FAST MULTIPLY

Most microprocessors offer a 16×16 or 32×32 bit multiply instruction. Multiply when performed sequentially takes one shift/add per bit, or 32 cycles for 32 bit data. The microprocessor **50** provides a high speed multiply which allows multiplication by small numbers using only a small number of cycles FIG. **23** shows the logic used to implement the high speed algorithm. To perform a multiply, the size of the multiplier less one is placed in the DOWN COUNTER **472**. For a four bit multiplier, the number three would be stored in the DOWN COUNTER **472**. Zero is loaded into the A register **474**. The multiplier is written bit reversed into the B Register **476** For example, a bit reversed five (binary 0101) would be written into B as 1010. The multiplicand is written into the C register **470** Executing the FAST MULT instruction will leave the result in the A Register **474**, when the count has been completed. The fast multiply instruction is important because many applications scale one number by a much smaller number The difference in speed between multiplying a 32×32 bit and a 32×4 bit is a factor of 8. If the least significant bit of the multiplier is a "ONE", the contents of the A register **474** and the C register **470** are added. If the least significant bit of the multiplier is a "ZERO", the contents of the A register are passed through the ALU **80** unaltered The output of the ALU **80** is shifted left by shifter **482** in each iteration. The contents of the B register **476** are shifted right by the shifter **480** in each iteration

INSTRUCTION EXECUTION PHILOSOPHY

The microprocessor **50** uses high speed D latches in most of the speed critical areas Slower on-chip RAM is used as secondary storage.

The microprocessor **50** philosophy of instruction execution is to create a hierarchy of speed as follows:

Logic and D latch transfers	1 cycle	20 nsec
Math	2 cycles	40 nsec
Fetch/store on-chip RAM	2 cycles	40 nsec
Fetch/store in current RAS page	4 cycles	80 nsec
Fetch/store with RAS cycle	11 cycles	220 nsec

With a 50 MHZ clock, many operations can be performed in 20 nsec and almost everything else in 40 nsec.

To maximize speed, certain techniques in processor design have been used. They include:

Eliminating arithmetic operations on addresses,

Fetching up to four instructions per memory cycle.

Pipelineless instruction decoding

Generating results before they are needed

Use of three level stack caching.

PIPELINE PHILOSOPHY

Computer instructions are usually broken down into sequential pieces, for example: fetch, decode, register read, execute, and store. Each piece will require a single machine cycle. In most Reduced Instruction Set Computer (RISC) chips, instruction require from three to six cycles

RISC instructions are very parallel. For example, each of 70 different instructions in the SPARC (SUN Computer's RISC chip) has five cycles. Using a technique called "pipelining", the different phases of consecutive instructions can be overlapped.

To understand pipelining think of building five residential homes. Each home will require in sequence, a foundation, framing, plumbing and wiring, roofing, and interior finish. Assume that each activity takes one week To build one house will take five weeks.

But what if you want to build an entire subdivision? You have only one of each work crew, but when the foundation men finish on the first house, you immediately start them on the second one, and so on At the end of five weeks, the first home is complete, but you also have five foundations. If you have kept the framing, plumbing, roofing, and interior guys all busy, from five weeks on, a new house will be completed each week

This is the way a RISC chip like SPARC appears to execute an instruction in a single machine cycle In reality, a RISC chip is executing one fifth of five instructions each machine cycle And if five instructions stay in sequence an instruction will be completed each machine cycle

The problems with a pipeline are keeping the pipe full with instructions. Each time an out of sequence instruction such as a BRANCH or CALL occurs, the pipe must be refilled with the next sequence The resulting dead time to refill the pipeline can become substantial when many IF/THEN/ELSE statements or subroutines are encountered.

THE PIPELINE APPROACH

The microprocessor **50** has no pipeline as such. The approach of this microprocessor to speed is to overlap instruction fetching with execution of the previously fetched instruction(s) Beyond that, over half the instructions (the most common ones) execute entirely in a single machine cycle of 20 nsec. This is possible because:

1 Instruction decoding resolves in 2 5 nsec

2 Incremented/decremented and some math values are calculated before they are needed, requiring only a latching signal to execute

3 Slower memory is hidden from high speed operations by high-speed D latches which access in 4 nsec.

The disadvantage for this microprocessor is a more complex chip design process. The advantage for the chip user is faster ultimate throughput since pipeline stalls cannot exist Pipeline synchronization with availability flag bits and other such pipeline handling is not required by this microprocessor.

For example, in some RISC machines an instruction which tests a status flag may have to wait for up to four cycles for the flag set by the previous instruction to be available to be tested Hardware and software debugging is also somewhat easier because the user doesn't have to visualize five instructions simultaneously in the pipe.

OVERLAPPING INSTRUCTION FETCH/EXECUTE

The slowest procedure the microprocessor **50** performs is to access memory. Memory is accessed when data is read or written. Memory is also read when instructions are fetched. The microprocessor **50** is able to hide fetch of the next instruction behind the execution of the previously fetched instruction(s) The microprocessor **50** fetches instructions in 4-byte instruction groups. An instruction group may contain from one to four instructions The amount of time required to execute the instruction group ranges from 4 cycles for simple instructions to 64 cycles for a multiply.

When a new instruction group is fetched, the microprocessor instruction decoder looks at the most significant bit of all four of the bytes. The most significant bit of an instruction determines if a memory access is required. For example, CALL, FETCH, and STORE all require a memory access to execute If all four bytes have nonzero most significant bits, the microprocessor initiates the memory fetch of the next sequential 4-byte instruction group. When the last instruction in the group finishes executing, the next 4-byte instruction group is ready and waiting on the data bus needing only to be latched into the instruction register If the 4-byte instruction group required four or more cycles to execute and the next sequential access was a column address strobe (CAS) cycle, the instruction fetch was completely overlapped with execution

INTERNAL ARCHITECTURE

The microprocessor **50** architecture consists of the following:

PARAMETER STACK	<--> ALU* <-->	Y REGISTER RETURN STACK
<--- 32 BITS --- >		<---32 BITS--->
16 DEEP		16 DEEP
Used for math and logic		Used for subroutine and interrupt return addresses as well as local variables
Push down stack. Can overflow into off-chip RAM		Push down stack Can overflow into off-chip RAM. Can also be accessed relative to top of stack
LOOP COUNTER		(32-bits. can decrement by 1) Used by class of test and loop instructions.
X REGISTER		(32-bits, can increment or decrement by 4). Used to point to RAM locations.
PROGRAM COUNTER		(32-bits, increments by 4). Points to 4-byte instruction groups in RAM
INSTRUCTION REG		(32-Bits) Holds 4-byte instruction groups while they are being decoded and executed

*Math and logic operations use the TOP item and NEXT to top Parameter Stack items as the operands. The result is pushed onto the Parameter Stack.
*Return addresses from subroutines are placed on the Return Stack. The Y REGISTER is used as a pointer to RAM locations. Since the Y REGISTER is the top item of the Return Stack nesting of indices is straightforward.
MODE - A register with mode and status bits
MODE-BITS:
Slow down memory accesses by 8 if "1". Run full speed if "0". (Provided for access to slow EPROM)
Divide the system clock by 1023 if "1" to reduce power consumption. Run full speed if "0" (On-chip counters slow down if this bit is set)
Enable external interrupt 1
Enable external interrupt 2
Enable external interrupt 3
Enable external interrupt 4
Enable external interrupt 5
Enable external interrupt 6
Enable external interrupt 7.
ON-CHIP MEMORY LOCATIONS.

MODE-BITS	
DMA-POINTER	
DMA-COUNTER	
STACK-POINTER	Pointer into Parameter Stack.
STACK-DEPTH	Depth of on-chip Parameter Stack
RSTACK-POINTER	Pointer into Return Stack
RSTACK-DEPTH	Depth of on-chip Return Stack

ADDRESSING MODE HIGH POINTS

The data bus is 32-bits wide. All memory fetches and stores are 32-bits. Memory bus addresses are 30 bits. The least significant 2 bits are used to select one-of-four bytes in some addressing modes. The Program Counter, X Register, and Y Register are implemented as D latches with their outputs going to the memory address bus and the bus incrementer/decrementer. Incrementing one of these registers can happen quickly because the incremented value has already rippled through the inc/dec logic and need only be clocked into the latch. Branches and Calls are made to 32-bit word-boundaries.

INSTRUCTION SET

32-BIT INSTRUCTION FORMAT

The thirty two bit instructions are CALL, BRANCH BRANCH-IF-ZERO and LOOP-IF-NOT-DONE. These instructions require the calculation of an effective address. In many computers, the effective address is calculated by adding or subtracting an operand with the current Program Counter. This math operation requires from four to seven machine cycles to perform and can definitely bog down machine execution. The microprocessor's strategy is to perform the required math operation at assembly or linking time and do a much simpler "Increment to next page" or "Decrement to previous page" operation at run time. As a result, the microprocessor branches execute in a single cycle.

24-BIT OPERAND FORM:
Byte 1 Byte 2 Byte 3 Byte 4
WWWWWW XX - YYYYYYYY - YYYYYYYY - YYYYYYYY
With a 24-bit operand, the current page is considered to be defined by the most significant 6 bits of the Program Counter
16-BIT OPERAND FORM:
QQQQQQQQ - WWWWWW XX - YYYYYYYY - YYYYYYYY
With a 16-bit operand, the current page is considered to be defined by the most significant 14 bits of the Program Counter
8-BIT OPERAND FORM:
QQQQQQQQ - QQQQQQQQ - WWWWWW XX - YYYYYYYY
With an 8-bit operand, the current page is considered to be defined by the most significant 22 bits of the Program Counter
QQQQQQQQ - Any 8-bit instruction
WWWWWW - Instruction op-code.
XX - Select how the address bits will be used:
00 - Make all high-order bits zero (Page zero addressing)
01 - Increment the high-order bits. (Use next page)
10 - Decrement the high-order bits (Use previous page)
11 - Leave the high-order bits unchanged. (Use current page)
YYYYYYYY - The address operand field. This field is always shifted left two bits (to generate a word rather than byte address) and loaded into the Program Counter. The microprocessor instruction decoder figures out the width of the operand field by the location of the instruction op-code in the four bytes.

The compiler or assembler will normally use the shortest operand required to reach the desired address so that the leading bytes can be used to hold other instructions. The effective address is calculated by combining:

The current Program Counter

The 8, 16, or 24 bit address operand in the instruction.

Using one of the four allowed addressing modes

EXAMPLES OF EFFECTIVE ADDRESS CALCULATION

Example 1:

Byte 1	Byte 2	Byte 3	Byte 4
QQQQQQQQ	QQQQQQQQ	00000011	10011000

The "QQQQQQQQs" in Byte **1** and **2** indicate space in the 4-byte memory fetch which could be hold two other instructions to be executed prior to the CALL instruction. Byte **3** indicates a CALL instruction (six zeros) in the current page (indicated by the 11 bits). Byte 4 indicates that the hexadecimal number 98 will be forced into the Program Counter bits **2** through **10** (Remember, a CALL or BRANCH always goes to a word boundary so the two least

significant bits are always set to zero). The effect of this instruction would be to CALL a subroutine at WORD location HEX 98 in the current page. The most significant 22 bits of the Program Counter define the current page and will be unchanged.

Example 2:

Byte 1	Byte 2	Byte 3	Byte 4
000001 01	00000001	00000000	00000000

If we assume that the Program Counter was HEX 0000 0156 which is binary:

00000000 00000000 00000001 01010110 = OLD PROGRAM COUNTER

Byte **1** indicates a BRANCH instruction op code (000001) and "01" indicates select the next page. Byte **2**,**3**, and 4 are the address operand. These 24-bits will be shifted to the left two places to define a WORD address. HEX 0156 shifted left two places is HEX 0558. Since this is a 24-bit operand instruction, the most significant 6 bits of the Program Counter define the current page. These six bits will be incremented to select the next page. Executing this instruction will cause the Program Counter to be loaded with HEX 0400 0558 which is binary:

00000100 00000000 00000101 01011000 = NEW PROGRAM COUNTER.

INSTRUCTIONS

CALL-LONG

0000 00XX - YYYYYYYY - YYYYYYYY - YYYYYYYY

Load the Program Counter with the effective WORD address specified. Push the current PC contents onto the RETURN STACK.

OTHER EFFECTS: CARRY or modes, no effect. May cause Return Stack to force an external memory cycle if on-chip Return Stack is full

BRANCH

0000 01XX - YYYYYYYY - YYYYYYYY - YYYYYYYY

Load the Program Counter with the effective WORD address specified.

OTHER EFFECTS: NONE

BRANCH-IF-ZERO

0000 10XX - YYYYYYYY - YYYYYYYY - YYYYYYYY

Test the TOP value on the Parameter Stack. If the value is equal to zero load the Program Counter with the effective WORD address specified. If the TOP value is not equal to zero, increment the Program Counter and fetch and execute the next instruction.

OTHER EFFECTS: NONE

LOOP-IF-NOT-DONE

0000 11YY - (XXXX XXXX) - (XXXX XXXX) - (XXXX XXXX)

If the LOOP COUNTER is not zero, load the Program Counter with the effective WORD address specified. If the LOOP COUNTER is zero, decrement the LOOP COUNTER, increment the Program Counter and fetch and execute the next instruction.

OTHER EFFECTS: NONE

8-BIT INSTRUCTIONS PHILOSOPHY

Most of the work in the microprocessor **50** is done by the 8-bit instructions. Eight bit instructions are possible with the microprocessor because of the extensive use of implied stack addressing. Many 32-bit architectures use 8-bits to specify the operation to perform but use an additional 24-bits to specify two sources and a destination.

For math and logic operations, the microprocessor **50** exploits the inherent advantage of a stack by designating the source operand(s) as the top stack item and the next stack item. The math or logic operation is performed, the operands are popped from the stack and the result is pushed back on the stack. The result is a very efficient utilization of instruction bits as well as registers. A comparable situation exists between Hewlett Packard calculators (which use a stack) and Texas Instrument calculators which don't. The identical operation on an HP will require one half to one third the keystrokes of the TI.

The availability of 8-bit instructions also allows another architectural innovation: the fetching of four instructions in a single 32-bit memory cycle. The advantages of fetching multiple instructions are:

Increased execution speed even with slow memories.

Similar performance to the Harvard (separate data and instruction busses) without the expense,

Opportunities to optimize groups of instructions,

The capability to perform loops within this mini-cache. The microloops inside the four instruction group are effective for searches and block moves.

SKIP INSTRUCTIONS

The microprocessor **50** fetches instructions in 32-bit chunks called 4-byte instruction groups. These four bytes may contain four 8-bit instructions or some mix of 8-bit and 16 or 24-bit instructions. SKIP instructions in the microprocessor skip any remaining instructions in a 4-byte instruction group and cause a memory fetch to get the next 4-byte instruction group. Conditional SKIPs when combined with 3-byte BRANCHES will create conditional BRANCHES. SKIPs may also be used in situations when no use can be made of the remaining bytes in a 4-instruction group. A SKIP executes in a single cycle whereas a group of three NOPs would take three cycles.

SKIP-ALWAYS -	skip any remaining instructions in this 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group.
SKIP-IF-ZERO -	If the TOP item of the Parameter Stack is zero, skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the TOP item is not zero, execute the next sequential instruction.
SKIP-IF-POSITIVE -	If the TOP item of the Parameter Stack has a the most significant bit (the sign bit) equal to "0", skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the TOP item is not "0", execute the next sequential instruction

-continued

SKIP-IF-NO-CARRY -	If the CARRY flag from a SHIFT or arithmetic operation is not equal to "1", skip any remaining instructions in the 4-byte instruction group Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group If the CARRY is equal to "1", execute the next sequential instruction.
SKIP-NEVER (NOP)	Execute the next sequential instruction (Delay one machine cycle).
SKIP-IF-NOT-ZERO -	If the TOP item on the Parameter Stack is not equal to "0", skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group If the TOP item is equal 0 , execute the next sequential instruction
SKIP-IF-NEGATIVE -	If the TOP item on the Parameter Stack has its most significant bit (sign bit) set to "1" skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the TOP item has its most significant bit set to "0", execute the next sequential instruction.
SKIP-IF-CARRY -	If the CARRY flag is set to 1 as a result of SHIFT or arithmetic operation, skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30-bits of the Program Counter and proceed to fetch the next 4-byte instruction group. If the CARRY flag is "0", execute the next sequential instruction

MICROLOOPS

Microloops are a unique feature of the microprocessor architecture which allows controlled looping within a 4-byte instruction group. A microloop instruction tests the LOOP COUNTER for "0" and may perform an additional test. If the LOOP COUNTER is not "0" and the test is met, instruction execution continues with the first instruction in the 4-byte instruction group, and the LOOP COUNTER is decremented. A microloop instruction will usually be the last byte in a 4-byte instruction group, but it can be any byte If the LOOP COUNTER is "0" or the test is not met instruction execution continues with the next instruction. If the microloop is the last byte in the 4-byte instruction group, the most significant 30-bits of the Program Counter are incremented and the next 4-byte instruction group is fetched from memory On a termination of the loop on LOOP COUNTER equal to "0", the LOOP COUNTER will remain at "0". Microloops allow short iterative work such as moves and searches to be performed without slowing down to fetch instructions from memory

EXAMPLE:

Byte 1 FETCH-VIA-X-AUTOINCREMENT	Byte 2 STORE-VIA-Y-AUTO-INCREMENT
Byte 3 ULOOP-UNTIL-DONE	Byte 4 QQQQQQQQ

This example will perform a block move To initiate the transfer, X will be loaded with the starting address of the source. Y will be loaded with the starting address of the destination. The LOOP COUNTER will be loaded with the number of 32-bit words to move The microloop will FETCH and STORE and count down the LOOP COUNTER until it reaches zero QQQQQQQQ indicates any instruction can follow.

MICROLOOP INSTRUCTIONS

ULOOP-UNTIL-DONE—If the LOOP COUNTER is not "0", continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0", continue execution with the next instruction

ULOOP-IF-ZERO—If the LOOP COUNTER is not "0" and the TOP item on the Parameter Stack is "0", continue execution with the first instruction in the 4-byte instruction group Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the TOP item is "1", continue execution with the next instruction.

ULOOP-IF-POSITIVE—If the LOOP COUNTER is not "0" and the most significant bit (sign bit) is "0", continue execution with the first instruction in the 4-byte instruction group Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the TOP item is "1", continue execution with the next instruction

ULOOP-IF-NOT-CARRY-CLEAR—If the LOOP COUNTER is not "0" and the floating point exponents found in TOP and NEXT are not aligned continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the exponents are aligned continue execution with the next instruction. This instruction is specifically designed for combination with special SHIFT instructions to align two floating point numbers.

ULOOP-NEVER—(DECREMENT-LOOP-COUNTER) Decrement the LOOP COUNTER. Continue execution with the next instruction.

ULOOP-IF-NOT-ZERO—If the LOOP COUNTER is not "0" and the TOP item of the Parameter Stack is "0" continue execution with the first instruction in the 4-byte instruction group Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the TOP item is "1", continue execution with the next instruction.

ULOOP-IF-NEGATIVE—If the LOOP COUNTER is not "0" and the most significant bit (sign bit) of the TOP item of the Parameter Stack is "1" continue execution with the first instruction in the 4-byte instruction group Decrement the LOOP COUNTER If the LOOP COUNTER is "0" or the most significant bit of the Parameter Stack is "0", continue execution with the next instruction.

ULOOP-IF-CARRY-SET—If the LOOP COUNTER is not "0" and the exponents of the floating point numbers found in TOP and NEXT are not aligned, continue execution with the first instruction in the 4-byte instruction group. Decrement the LOOP COUNTER. If the LOOP COUNTER is "0" or the exponents are aligned, continue execution with the next instruction.

RETURN FROM SUBROUTINE OR INTERRUPT

Subroutine calls and interrupt acknowledgements cause a redirection of normal program execution. In both cases, the current Program Counter is pushed onto the Return Stack, so the microprocessor can return to its place in the program after executing the subroutine or interrupt service routine

NOTE: When a CALL to subroutine or interrupt is acknowledged the Program Counter has already been incremented and is pointing to the 4-byte instruction group following the 4-byte group currently being executed. The instruction decoding logic allows the microprocessor to

perform a test and execute a return conditional on the outcome of the test in a single cycle A RETURN pops an address from the Return Stack and stores it to the Program Counter.

RETURN INSTRUCTIONS	
RETURN-ALWAYS -	Pop the top item from the Return Stack and transfer it to the Program Counter
RETURN-IF-ZERO -	If the TOP item on the Parameter Stack is "0", pop the top item from the Return Stack and transfer it to the Program Counter Otherwise execute the next instruction.
RETURN-IF-POSITIVE -	If the most significant bit (sign bit) of the TOP item on the Parameter Stack is a "0", pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction
RETURN-IF-CARRY-CLEAR -	If the exponents of the floating point numbers found in TOP and NEXT are not aligned pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.
RETURN-NEVER - (NOP)	Execute the next instruction
RETURN-IF-NOT-ZERO -	If the TOP item on the Parameter Stack is not "0", pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction
RETURN-IF-NEGATIVE -	If the most significant bit (sign bit) of the TOP item on the Parameter Stack is a "1" pop the top item from the Return Stack and transfer it to the Program Counter. Otherwise execute the next instruction.
RETURN-IF-CARRY-SET -	If the exponents of the floating point numbers found in TOP and NEXT are aligned, pop the top item from the Return Stack and transfer it to the Program. Counter. Otherwise execute the next instruction

HANDLING MEMORY FROM DYNAMIC RAM

The microprocessor **50**, like any RISC type architecture, is optimized to handle as many operations as possible on-chip for maximum speed External memory operations take from 80 nsec. to 220 nsec compared with on-chip memory speeds of from 4 nsec. to 30 nsec There are times when external memory must be accessed

External memory is accessed using three registers:

X-REGISTER—A 30-bit memory pointer which can be used for memory access and simultaneously incremented or decremented.

Y-REGISTER—A 30-bit memory pointer which can be used for memory access and simultaneously incremented or decremented.

PROGRAM-COUNTER—A 30-bit memory pointer normally used to point to 4-byte instruction groups External memory may be accessed at addresses relative to the PC. The operands are sometimes called "Immediate' or "Literal" in other computers When used as memory pointer, the PC is also incremented after each operation.

MEMORY LOAD & STORE INSTRUCTIONS

FETCH-VIA-X—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. X is unchanged

FETCH-VIA-Y—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack Y is unchanged.

FETCH-VIA-X-AUTOINCREMENT—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack After fetching, increment the most significant 30 bits of X to point to the next 32-bit word address.

FETCH-VIA-Y-AUTOINCREMENT—Fetch the 32-bit memory content pointed to by Y and push it onto the Parameter Stack After fetching, increment the most significant 30 bits of Y to point to the next 32-bit word address.

FETCH-VIA-X-AUTODECREMENT—Fetch the 32-bit memory content pointed to by X and push it onto the Parameter Stack. After fetching, decrement the most significant 30 bits of X to point to the previous 32-bit word address

FETCH-VIA-Y-AUTODECREMENT—Fetch the 32-bit memory content pointed to by Y and push it onto the Parameter Stack After fetching, decrement the most significant 30 bits of Y to point to the previous 32-bit word address

STORE-VIA-X—Pop the top item of the Parameter Stack and store it in the memory location pointed to by X X is unchanged.

STORE-VIA-Y—Pop the top item of the Parameter Stack and store it in the memory location pointed to by Y. Y is unchanged

STORE-VIA-X-AUTOINCREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by X After storing, increment the most significant 30 bits of X to point to the next 32-bit word address.

STORE-VIA-Y-AUTOINCREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by Y After storing, increment the most significant 30 bits of Y to point to the next 32-bit word address.

STORE-VIA-X-AUTODECREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by X. After storing, decrement the most significant 30 bits of X to point to the previous 32-bit word address.

STORE-VIA-Y-AUTODECREMENT—Pop the top item of the Parameter Stack and store it in the memory location pointed to by Y After storing decrement the most significant 30 bits of Y to point to the previous 32-bit word address.

FETCH-VIA-PC—Fetch the 32-bit memory content pointed to by the Program Counter and push it onto the Parameter Stack After fetching, increment the most significant 30 bits of the Program Counter to point to the next 32-bit word address.

*NOTE When this instruction executes, the PC is pointing to the memory location following the instruction. The effect is of loading a 32-bit immediate operand This is an 8-bit instruction and therefore will be combined with other 8-bit instructions in a 4-byte instruction fetch It is possible to have from one to four FETCH-VIA-PC instructions in a 4-byte instruction fetch. The PC incre-

ments after each execution of FETCH-VIA-PC, so it is possible to push four immediate operands on the stack. The four operands would be the found in the four memory locations following the instruction.

BYTE-FETCH-VIA-X—Fetch the 32-bit memory content pointed to by the most significant 30 bits of X. Using the two least significant bits of X, select one of four bytes from the 32-bit memory fetch, right justify the byte in a 32-bit field and push the selected byte preceded by leading zeros onto the Parameter Stack.

BYTE-STORE-VIA-X—Fetch the 32-bit memory content pointed to by the most significant 30 bits of X. Pop the TOP item from the Parameter Stack. Using the two least significant bits of X place the least significant byte into the 32-bit memory data and write the 32-bit entity back to the location pointed to by the most significant 30 bits of X.

OTHER EFFECTS OF MEMORY ACCESS INSTRUCTIONS:

Any FETCH instruction will push a value on the Parameter Stack **74**. If the on-chip stack is full, the stack will overflow into off-chip memory stack resulting in an additional memory cycle. Any STORE instruction will pop a value from the Parameter Stack **74**. If the on-chip stack is empty, a memory cycle will be generated to fetch a value from off-chip memory stack.

HANDLING ON-CHIP VARIABLES

High-level languages often allow the creation of LOCAL VARIABLES. These variables are used by a particular procedure and discarded. In cases of nested procedures layers of these variables must be maintained. On-chip storage is up to five times faster than off-chip RAM, so a means of keeping local variables on-chip can make operations run faster. The microprocessor **50** provides the capability for both on-chip storage of local variables and nesting of multiple levels of variables through the Return Stack.

The Return Stack **134** is implemented as 16 on-chip RAM locations. The most common use for the Return Stack **134** is storage of return addresses from subroutines and interrupt calls. The microprocessor allows these 16 locations to also be used as addressable registers. The 16 locations may be read and written by two instructions which indicate a Return Stack relative address from 0–15. When high-level procedures are nested, the current procedure variables push the previous procedure variables further down the Return Stack **134**. Eventually, the Return Stack will automatically overflow into off-chip RAM.

ON-CHIP VARIABLE INSTRUCTIONS

READ-LOCAL-VARIABLE XXXX—Read the XXXXth location relative to the top of the Return Stack. (XXXX is a binary number from 0000–1111). Push the item read onto the Parameter Stack.

OTHER EFFECTS: If the Parameter Stack is full, the push operation will cause a memory cycle to be generated as one item of the stack is automatically stored to external RAM. The logic which selects the location performs a modulo 16 subtraction. If four local variables have been pushed onto the Return Stack, and an instruction attempts to READ the fifth item, unknown data will be returned.

WRITE-LOCAL-VARIABLE XXXX—Pop the TOP item of the Parameter Stack and write it into the XXXXth location relative to the top of the Return Stack. (XXXX is a binary number from 0000–1111.)

OTHER EFFECTS: If the Parameter Stack is empty, the pop operation will cause a memory cycle to be generated to fetch the Parameter Stack item from external RAM. The logic which selects the location performs a modulo 16 subtraction. If four local variables have been pushed onto the Return Stack and an instruction attempts to WRITE to the fifth item, it is possible to clobber return addresses or wreak other havoc.

REGISTER AND FLIP-FLOP TRANSFER AND PUSH INSTRUCTIONS

DROP—Pop the TOP item from the Parameter Stack and discard it.

SWAP—Exchange the data in the TOP Parameter Stack location with the data in the NEXT Parameter Stack location.

DUP—Duplicate the TOP item on the Parameter Stack and push it onto the Parameter Stack.

PUSH-LOOP-COUNTER—Push the value in LOOP COUNTER onto the Parameter Stack.

POP-RSTACK-PUSH-TO-STACK—Pop the top item from the Return Stack and push it onto the Parameter Stack.

PUSH-X-REG—Push the value in the X Register onto the Parameter Stack.

PUSH-STACK-POINTER—Push the value of the Parameter Stack pointer onto the Parameter Stack.

PUSH-RSTACK-POINTER—Push the value of the Return Stack pointer onto the Return Stack.

PUSH-MODE-BITS—Push the value of the MODE REGISTER onto the Parameter Stack.

PUSH-INPUT—Read the 10 dedicated input bits and push the value (right justified and padded with leading zeros) onto the Parameter Stack.

SET-LOOP-COUNTER—Pop the TOP value from the Parameter Stack and store it into LOOP COUNTER.

POP-STACK-PUSH-TO-RSTACK—Pop the TOP item from the Parameter Stack and push it onto the Return Stack.

SET-X-REG—Pop the TOP item from the Parameter Stack and store it into the X Register.

SET-STACK-POINTER—Pop the TOP item from the Parameter Stack and store it into the Stack Pointer.

SET-RSTACK-POINTER—Pop the TOP item from the Parameter Stack and store it into the Return Stack Pointer.

SET-MODE-BITS—Pop the TOP value from the Parameter Stack and store it into the MODE BITS.

SET-OUTPUT—Pop the TOP item from the Parameter Stack and output it to the 10 dedicated output bits.

OTHER EFFECTS: Instructions which push or pop the Parameter Stack or Return Stack may cause a memory cycle as the stacks overflow back and forth between on-chip and off-chip memory.

LOADING A SHORT LITERAL

A special case of register transfer instruction is used to push an 8-bit literal onto the Parameter Stack. This instruction requires that the 8-bits to be pushed reside in the last byte of a 4-byte instruction group. The instruction op-code loading the literal may reside in ANY of the other three bytes in the instruction group.

EXAMPLE:

BYTE 1	BYTE 2	BYTE 3
LOAD-SHORT-LITERAL	QQQQQQQQ	QQQQQQQQ
BYTE 4		
00001111		

In this example, QQQQQQQQ indicates any other 8-bit instruction. When Byte **1** is executed, binary 00001111 (HEX 0f) from Byte **4** will be pushed (right justified and padded by leading zeros) onto the Parameter Stack. Then the instructions in Byte **2** and Byte **3** will execute. The microprocessor instruction decoder knows not to execute Byte **4**. It is possible to push three identical 8-bit values as follows:

BYTE 1	BYTE 2
LOAD-SHORT-LITERAL	LOAD-SHORT-LITERAL
BYTE 3	BYTE 4
LOAD-SHORT-LITERAL	00001111
SHORT-LITERAL-INSTRUCTION	
LOAD-SHORT-LITERAL -	Push the 8-bit value found in Byte 4 of the current 4-byte instruction group onto the Parameter Stack

LOGIC INSTRUCTIONS

Logical and math operations used the stack for the source of one or two operands and as the destination for results The stack organization is a particularly convenient arrangement for evaluating expressions. TOP indicates the top value on the Parameter Stack **74**. NEXT indicates the next to top value on the Parameter Stack **74**

AND—Pop TOP and NEXT from the Parameter Stack, perform the logical AND operation on these two operands, and push the result onto the Parameter Stack

OR—Pop TOP and NEXT from the Parameter Stack, perform the logical OR operation on these two operands, and push the result onto the Parameter Stack

XOR—Pop TOP and NEXT from the Parameter Stack perform the logical exclusive OR on these two operands, and push the result onto the Parameter Stack.

BIT-CLEAR—Pop TOP and NEXT from the Parameter Stack, toggle all bits in NEXT, perform the logical AND operation on TOP, and push the result onto the Parameter Stack (Another way of understanding this instruction is thinking of it as clearing all bits in TOP that are set in NEXT.)

MATH INSTRUCTIONS

Math instruction pop the TOP item and NEXT to top item of the Parameter Stack **74** to use as the operands. The results are pushed back on the Parameter Stack The CARRY flag is used to latch the "33rd bit" of the ALU result

ADD—Pop the TOP item and NEXT to top item from the Parameter Stack, add the values together and push the result back on the Parameter Stack The CARRY flag may be changed.

ADD-WITH-CARRY—Pop the TOP item and the NEXT to top item from the Parameter Stack, add the values together. If the CARRY flag is "1" increment the result. Push the ultimate result back on the Parameter Stack. The CARRY flag may be changed.

ADD-X—Pop the TOP item from the Parameter Stack and read the third item from the top of the Parameter Stack. Add the values together and push the result back on the Parameter Stack. The CARRY flag may be changed

SUB—Pop the TOP item and NEXT to top item from the Parameter Stack, Subtract NEXT from TOP and push the result back on the Parameter Stack The CARRY flag may be changed.

SUB-WITH-CARRY—Pop the TOP item and NEXT to top item from the Parameter Stack Subtract NEXT from TOP If the CARRY flag is "1" increment the result. Push the ultimate result back on the Parameter Stack. The CARRY flag may be changed

SUB-X—

SIGNED-MULT-STEP—

UNSIGNED-MULT-STEP—

SIGNED-FAST-MULT—

FAST-MULT-STEP—

UNSIGNED-DIV-STEP—

GENERATE-POLYNOMIAL

ROUND—

COMPARE—Pop the TOP item and NEXT to top item from the Parameter Stack. Subtract NEXT from TOP If the result has the most significant bit equal to '0' (the result is positive), push the result onto the Parameter Stack. If the result has the most significant bit equal to "1" (the result is negative), push the old value of TOP onto the Parameter Stack The CARRY flag may be affected

SHIFT/ROTATE

SHIFT-LEFT—Shift the TOP Parameter Stack item left one bit. The CARRY flag is shifted into the least significant bit of TOP.

SHIFT-RIGHT—Shift the TOP Parameter Stack item right one bit. The least significant bit of TOP is shifted into the CARRY flag Zero is shifted into the most significant bit of TOP.

DOUBLE-SHIFT-LEFT—Treating the TOP item of the Parameter Stack as the most significant word of a 64-bit number and the NEXT stack item as the least significant word, shift the combined 64-bit entity left one bit The CARRY flag is shifted into the least significant bit of NEXT.

DOUBLE-SHIFT-RIGHT—Treating the TOP item of the Parameter Stack as the most significant word of a 64-bit number and the NEXT stack item as the least significant word, shift the combined 64-bit entity right one bit The least significant bit of NEXT is shifted into the CARRY flag. Zero is shifted into the most significant bit of TOP.

OTHER INSTRUCTIONS

FLUSH-STACK—Empty all on-chip Parameter Stack locations into off-chip RAM (This instruction is useful for multitasking applications). This instruction accesses a counter which holds the depth of the on-chip stack and can require from none to 16 external memory cycles.

FLUSH-RSTACK—Empty all on-chip Return Stack locations into off-chip RAM. (This instruction is useful for multitasking applications). This instruction accesses a counter which holds the depth of the on-chip Return Stack and can require from none to 16 external memory cycles

It should further be apparent to those skilled in the art that various changes in form and details of the invention as shown and described may be made It is intended that such changes be included within the spirit and scope of the claims appended hereto

What is claimed is:

1. A microprocessor, which comprises a main central processing unit and a separate direct memory access central processing unit in a single integrated circuit comprising said microprocessor, said main central processing unit having an arithmetic logic unit, a first push down stack with a top item register and a next item register, connected to provide inputs to said arithmetic logic unit, an output of said arithmetic logic unit being connected to said top item register, said top item register also being connected to provide inputs to an internal data bus, said internal data bus being bidirectionally connected to a loop counter, said loop counter being connected to a decrementer, said internal data bus being bidirectionally connected to a stack pointer, return stack pointer, mode register and instruction register, said internal data bus being connected to a memory controller, to a Y register of a return push down stack, an X register and a program counter, said Y register, X register and program counter providing outputs to an internal address bus, said internal address bus providing inputs to said memory controller and to an incrementer, said incrementer being connected to said internal data bus, said direct memory access central processing unit providing inputs to said memory controller, said memory controller having an address/data bus and a plurality of control lines for connection to a random access memory.

2. The microprocessor of claim 1 in which said memory controller includes a multiplexing means between said central processing unit and said address/data bus, said multiplexing means being connected and configured to provide row addresses, column addresses and data on said address/data bus

3. The microprocessor of claim 1 in which said memory controller includes means for fetching instructions for said central processing unit on said address/data bus, said means for fetching instructions being configured to fetch multiple sequential instructions in a single memory cycle

4. The microprocessor of claim 3 additionally comprising means connected to said means for fetching instructions for determining if multiple instructions fetched by said means for fetching instructions require a memory access, said means for fetching instructions fetching additional multiple instructions if the multiple instructions do not require a memory access

5. The microprocessor of claim 3 in which said microprocessor and a dynamic random access memory are contained in a single integrated circuit and said means for fetching instructions includes a column latch for receiving the multiple instructions

6. The microprocessor of claim 1 in which said microprocessor includes a sensing circuit and a driver circuit, and an output enable line for connection between the random access memory, said sensing circuit and said driver circuit, said sensing circuit being configured to provide a ready signal when said output enable line reaches a predetermined electrical level, said microprocessor being configured so that said driver circuit provides an enabling signal on said output enable line responsive to the ready signal

7. The microprocessor of claim 1 additionally comprising a ring oscillator variable speed system clock connected to said main central processing unit, said main central processing unit and said ring oscillator variable speed system clock being provided in a single integrated circuit.

8. The microprocessor of claim 7 in which said memory controller includes an input/output interface connected to exchange coupling control signals addresses and data with said main central processing unit, said microprocessor additionally including a second clock independent of said ring oscillator variable speed system clock connected to said input/output interface

9. The microprocessor of claim 1 in which said first push down stack has a first plurality of stack elements configured as latches, a second plurality of stack elements configured as a random access memory said first and second plurality of stack elements and said central processing unit being provided in a single integrated circuit, and a third plurality of stack elements configured as a random access memory external to said single integrated circuit.

10. The microprocessor of claim 9 additionally comprising a first pointer connected to said first plurality of stack elements, a second pointer connected to said second plurality of stack elements, and a third pointer connected to said third plurality of stack elements, said central processing unit being connected to pop items from said first plurality of stack elements, said first stack pointer being connected to said second stack pointer to pop a first plurality of items from said second plurality of stack elements when said first plurality of stack elements are empty from successive pop operations by said central processing unit, said second stack pointer being connected to said third stack pointer to pop a second plurality of items from said third plurality of stack elements when said second plurality of stack elements are empty from successive pop operations by said central processing unit

* * * * *

The JS 44 civil cover sheet and the information contained herein neither replace nor supplement the filing and service of pleadings or other papers as required by law, except as provided by local rules of court This form, approved by the Judicial Conference of the United States in September 1974, is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet (SEE INSTRUCTIONS ON THE REVERSE OF THE FORM)

2-08 CV-226

I. (a) PLAINTIFFS

Technology Properties Limited and Patriot Scientific Corporation

(b) County of Residence of First Listed Plaintiff _____ (EXCEPT IN U S PLAINTIFF CASES)

(C) Attorney's (Firm Name, Address, and Telephone Number)

S Calvin Capshaw
Capshaw DeRieux, LLP
1127 Judson Road, Suite 220
P O Box 3999 (75606-3999)
Longview, TX 75601
(903) 236-9800

DEFENDANTS
HTC Corporation and HTC America, Inc.

County of Residence of First Listed Defendant _____ (IN U.S. PLAINTIFF CASES ONLY)

NOTE: IN LAND CONDEMNATION CASES USE THE LOCATION OF THE LAND INVOLVED

Attorneys (If Known)

II. BASIS OF JURISDICTION (PLACE AN 'X' IN ONE BOX ONLY)

1 U S. Government Plaintiff
☒ 3 Federal Question (U S Government Not a Party)
2 U S. Government Defendant
4 Diversity (Indicate Citizenship of Parties in Item III)

III. CITIZENSHIP OF PRINCIPAL PARTIES (Place an "X" in One Box for Plaintiff and One Box for Defendant) (For Diversity Cases Only)

	PTF	DEF		PTF	DEF
Citizen of This State	1	1	Incorporated or Principal Place of Business in This State	4	4
Citizen of Another State	2	2	Incorporated and Principal Place of Business in Another State	5	5
Citizen or Subject of a Foreign Nation	3	3	Foreign Country	6	6

IV. NATURE OF SUIT (Place an 'X' in One Box Only)

CONTRACT
110 Insurance
120 Marine
130 Miller Act
140 Negotiable Instrument
150 Recovery of Overpayment & Enforcement of Judgment
151 Medicare Act
152 Recovery of Defaulted Student Loans (Excl Veterans)
153 Recovery of Overpayment of Veteran's Benefits
160 Stockholders' Suits
190 Other Contract
195 Contract Product Liability
196 Franchise

TORTS
PERSONAL INJURY
310 Airplane
315 Airplane Product Liability
☐ 320 Assault Libel & Slander
330 Federal Employers Liability
☐ 340 Marine
345 Marine Product Liability
350 Motor Vehicle
355 Motor Vehicle Product Liability
360 Other Personal Injury
PERSONAL INJURY
362 Personal Injury - Med. Malpractice
365 Personal Injury – Product Liability
368 Asbestos Personal Injury Product Liability
PERSONAL PROPERTY
370 Other Fraud
371 Truth in Lending
380 Other Personal Property Damage
385 Property Damage Product Liability

REAL PROPERTY
210 Land Condemnation
220 Foreclosure
230 Rent Lease & Ejectment
240 Torts to Land
245 Tort Product Liability
290 All Other Real Property

CIVIL RIGHTS
441 Voting
442 Employment
443 Housing Accommodations
444 Welfare
440 Other Civil Rights

PRISONER PETITIONS
510 Motions to Vacate Sentence
HABEAS CORPUS:
530 General
535 Death Penalty
540 Mandamus & Other
550 Civil Rights
555 Prison Condition

FORFEITURE/PENALTY
610 Agriculture
620 Other Food & Drug
625 Drug Related Seizure of Property 21 USC 881
630 Liquor Laws
640 R R & Truck
650 Airline Regs
660 Occupational Safety/Health
690 Other

LABOR
710 Fair Labor Standards Act
720 Labor/Mgmt. Relations
730 Labor/Mgmt. Reporting & Disclosure Act
740 Railway Labor Act
790 Other Labor Litigation
791 Empl. Ret. Inc. Security Act

BANKRUPTCY
422 Appeal 28 USC 158
423 Withdrawal 28 USC 157

PROPERTY RIGHTS
820 Copyrights
X 830 Patent
840 Trademark

SOCIAL SECURITY
861 HIA (1 395ff)
862 Black Lung (923)
863 DIWC/DIWW (405(g))
864 SSID Title XVI
865 RSI (405(g))

FEDERAL TAX SUITS
870 Taxes (U.S. Plaintiff or Defendant
871 IRS – Third Party 26 USC 7609

OTHER STATUTES
400 State Reapportionment
410 Antitrust
430 Banks and Banking
450 Commerce
460 Deportation
470 Racketeer Influenced and Corrupt Organizations
480 Consumer Credit
490 Cable/Sat TV
810 Selective Service
850 Securities/Commodities/ Exchange
875 Customer Challenge 12 USC 3410
890 Other Statutory Actions
891 Agricultural Acts
892 Economic Stabilization Act
893 Environmental Matters
894 Energy Allocation Act
895 Freedom of Information Act
☐ 900 Appeal of Fee Determination Under Equal Access to Justice
950 Constitutionality of State State Statutes

V. ORIGIN (Place an "X" in One Box Only)

☒ 1 Original Proceeding | 2 Removed from State Court | 3 Remanded from Appellate Court | 4 Reinstated or Reopened | 5 Transferred from another district (specify) | 6 Multidistrict Litigation | 7 Appeal to District Judge from Magistrate Judgment

VI. CAUSE OF ACTION (Cite the U.S Civil Statute Under which you are filing (Do not cite jurisdictional statutes unless diversity): **35 U.S.C. § 271**
(Brief description of cause:

VII. REQUESTED IN COMPLAINT: CHECK IF THIS IS A CLASS ACTION UNDER FR C P 23 DEMAND $ CHECK YES only if demanded in complaint: JURY DEMAND: ☒ YES NO

VII. RELATED CASE(S) (See Instructions): **IF ANY** See the attached page.

DATE June 4, 2008 SIGNATURE OF ATTORNEY OF RECORD

FOR OFFICE USE ONLY

RECEIPT # _____ AMOUNT _____ APPLYING IFP _____ JUDGE _____ MAG JUDGE _____

Related Cases

1 Technology Properties Limited, Inc , et al v. HTC Corporation, et al; In the United States District Court for the Eastern District of Texas, Marshall Division - Cause No. 2:08-cv-172 (DF).

2. Technology Properties Limited, Inc., et al v ASUSTek Computer, Inc ; In the United States District Court for the Eastern District of Texas, Marshall Division - Cause No. 2:08-cv-177 (TJW)

3 Technology Properties Limited, Inc., et al v. Acer, Inc , et al; In the United States District Court for the Eastern District of Texas, Marshall Division; Cause No. 2:08-cv-176 (TJW).

4. Technology Properties Limited, Inc , et al v. HTC Corporation, et al; In the United States District Court for the Eastern District of Texas, Marshall Division - Cause No. 2:08-cv-_____ - No cause number yet – filed simultaneously on June 4, 2008.

5 Technology Properties Limited, Inc., et al v. Acer, Inc., et al; In the United States District Court for the Eastern District of Texas, Marshall Division - Cause No. 2:08-cv-_____ - No cause number yet – filed simultaneously on June 4, 2008

6. Technology Properties Limited, Inc., et al v. ASUSTek Computer, Inc ; In the United States District Court for the Eastern District of Texas, Marshall Division - Cause No 2:08-cv-_____ - No cause number yet – filed simultaneously on June 4, 2008

Exhibit H

E-Filed 6/17/08

IN THE UNITED STATES DISTRICT COURT

FOR THE NORTHERN DISTRICT OF CALIFORNIA

SAN JOSE DIVISION

MICRON TECHNOLOGY, INC.,

Plaintiff,

v.

MOSAID TECHNOLOGIES, INC.,

Defendant.

Case Number C 06-4496 JF (RS)

ORDER[1] DENYING MOTION TO TRANSFER

[re: docket no.43]

Defendant Mosaid Technologies, Inc. ("Mosaid") moves to transfer the instant action to the Eastern District of Texas pursuant to 28 U.S.C. § 1404(a). The Court has considered the briefing submitted by the parties and amicus curiae as well as the oral arguments presented at the hearing on May 30, 2008. For the reasons discussed below, the motion will be denied.

I. BACKGROUND

Micron Technology Inc. ("Micron") filed this action for declaratory relief against Mosaid on July 24, 2006. The next day, Mosaid filed a patent infringement suit in the Eastern District of Texas against Micron and Powerchip Semiconductor Corporation ("Powerchip"). On July 27,

[1] This disposition is not designated for publication in the official reporter.

2006, Mosaid moved to dismiss Micron's declaratory relief action for lack of subject matter jurisdiction. After a hearing on October 20, 2007, the Court granted Mosaid's motion.

Following an intervening change in the controlling law, the United States Court of Appeals for the Federal Circuit reversed and remanded on February 29, 2008. *Micron Technology, Inc. v. Mosaid Technologies, Inc.*, 518 F.3d 897 (Fed. Cir. 2008). The Federal Circuit also applied 28 U.S.C. § 1404(a) to determine the proper forum for the instant case and concluded that the Northern District of California is the more appropriate forum. *Micron*, 518 F.3d at 901, 902–03, 905. The Federal Circuit considered several "'convenience factors'" as part of its analysis: "the convenience and availability of witnesses, the absence of jurisdiction over all necessary or desirable parties, and the possibility of consolidation with related litigation." *Id.* at 905 (citing *Genentech* 998 F.2d at 938). The court found that neither the availability of witnesses nor jurisdiction over parties weighed in favor of either forum, and "the record [did] not show any ongoing litigation requiring consolidation." *Id.* Notably, the Federal Circuit stated "it would be an abuse of discretion to transfer the action." *Id.* Mosaid's motion for rehearing subsequently was denied. On April 23, 2008, Mosaid brought the instant motion to transfer the instant case for consolidation with the concurrently pending action in the Eastern District of Texas.

II. DISCUSSION

Section 1404(a) provides that "for the convenience of parties and witnesses, in the interest of justice, a district court may transfer any civil action to any other district or division where it might have been brought." 28 U.S.C. § 1404(a). As noted by the Federal Circuit, "the general rule favors the forum of the first-filed action" but "the trial courts have discretion to make exceptions to this general rule in the interest of justice or expediency." *Micron*, 518 F.3d at 904 (citing *Genentech Inc. v. Eli Lilly & Co.*, 998 F.2d 931, 937 (Fed. Cir. 1993)).

Mosaid argues that notwithstanding the unambiguous direction in the Federal Circuit's opinion, that court's § 1404(a) analysis has been superseded by subsequent developments in the litigation in the Eastern District of Texas. Because the scope of the Texas action is broader than that of the instant case—including additional patents and an additional defendant—and because

that action has progressed further toward trial, Mosaid claims that judicial efficiency now weighs in favor of transferring the instant case to the Eastern District of Texas notwithstanding the Federal Circuit's reasoning.

However, having conferred with the assigned judge in the Texas action, this Court concludes that the Texas action is not so advanced as to justify revisiting the Federal Circuit's § 1404 analysis. While the parties have conducted significant discovery and have resolved certain procedural issues, the Texas court has not conducted a *Markman* hearing. Both the Texas court and this Court have substantial experience adjudicating patent cases, and it appears that any discovery taken in the Texas action is transferable to this Court.

Micron asserts fourteen patents in the instant case; in the Texas action, Mosaid has asserted infringement by Micron of eight of these fourteen patents as well as infringement of four additional patents. With the exception of one patent not asserted against Powerchip, Mosaid has asserted identical claims against Micron and Powerchip. Because different patents are at issue in both actions, the parties will have to dedicate additional time and resources if the cases are consolidated irrespective of the forum in which the consolidated case ultimately is tried. Moreover, Powerchip has moved to intervene in the instant action, which further weakens Mosaid's argument that the Eastern District of Texas is the more appropriate forum because it is the only forum in which all patents are before the court.

Because the Court concludes that the circumstances have not changed significantly since the Federal Circuit conducted its § 1404 analysis, the motion to transfer will be denied. In keeping with its comments during oral argument, the Court will use its best efforts to expedite claim construction and the setting of an early trial date.

IV. ORDER

For the reasons set forth above, the motion to transfer is DENIED.

IT IS SO ORDERED.

DATED: June 17, 2008.

JEREMY FOGEL
United States District Judge

This Order has been served upon the following persons:

Gregory S. Arovas garovas@kirkland.com

Lina M. Brenner lmbrenner@duanemorris.com, bmcoffey@duanemorris.com

Henry C. Bunsow bunsowh@howrey.com, lim@howrey.com

Korula T. Cherian cheriank@howrey.com, lim@howrey.com

Sean DeBruine sdebruine@akingump.com, btseng@akingump.com, sbrowder@akingump.com, skapralov@akingump.com, westdocketing@akingump.com

Robert E. Freitas rfreitas@orrick.com, marlantico@orrick.com

Dan C. Hu hu@tphm.com

Jonathan M. James jjames@perkinscoie.com, cmason@perkinscoie.com, dgraziano@perkinscoie.com, docketphx@perkinscoie.com

Anthony S. Kim kimt@Howrey.com, guthartg@howrey.com

Bao Thuc Nguyen bnguyen@kirkland.com

Diana Marie Sangalli dsangalli@tphm.com, mbliler@tphm.com

Thomas W Sankey twsankey@duanemorris.com

Jeannine Yoo Sano sanoj@howrey.com

Kimberly Anne Schmitt kschmitt@kirkland.com

Stefani Elise Shanberg sshanberg@perkinscoie.com, sshanberg@wsgr.com, fgarcia@wsgr.com, shuckaby@wsgr.com

Michael C. Spillner mspillner@orrick.com, mortiz@orrick.com

Christian Chadd Taylor ctaylor@kirkland.com, jcsmith@kirkland.com

Robert Scott Wales WalesS@howrey.com, kalaheled@howrey.com

Kenneth Brian Wilson kwilson@perkinscoie.com, bollberding@perkinscoie.com, docketsflit@perkinscoie.com, lbailey@perkinscoie.com

David M. LaSpaluto
Perkins Coie Brown & Bain P.A.
2901 North Central Avenue
P.O. Box 400
Phoenix, AZ 85001-0400

Christopher M. Schultz
Perkins Coie Brown & Bain P.A.
2901 North Central Avenue
P.O. Box 400
Phoenix, AZ 85001-0400

UNITED STATES DISTRICT COURT
Northern District of California
450 Golden Gate Avenue
San Francisco, California 94102

www.cand.uscourts.gov

Richard W. Wieking
Clerk

General Court Number
415.522.2000

July 31, 2006

CASE NUMBER: CV 06-04496 JCS
CASE TITLE: MICRON TECHNOLOGY, INC.-v-MOSAID TECHNOLOGIES INCORPORATED

REASSIGNMENT ORDER

GOOD CAUSE APPEARING THEREFOR,

IT IS ORDERED that this case is reassigned to the **San Francisco** division.

Honorable WILLIAM H. ALSUP for all further proceedings.

Counsel are instructed that all future filings shall bear the initials **WHA** immediately after the case number.

ALL MATTERS PRESENTLY SCHEDULED FOR HEARING ARE VACATED AND SHOULD BE RENOTICED FOR HEARING BEFORE THE JUDGE TO WHOM THE CASE HAS BEEN REASSIGNED.

Date: 7/31/06

FOR THE EXECUTIVE COMMITTEE:

Richard W. Wieking
Clerk

NEW CASE FILE CLERK:

Copies to: Courtroom Deputies
Log Book Noted

Special Projects
Entered in Computer 7/31/06AS

CASE SYSTEMS ADMINISTRATOR:
Copies to: All Counsel

Transferor CSA